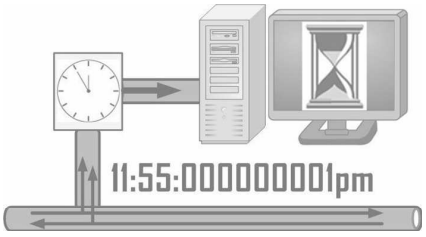
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2008

João Pedro Puga Faria

Sniffer para Redes Ethernet de Tempo-Real Baseado em FPGA



João Pedro Puga Faria

Sniffer para Redes Ethernet de Tempo-Real Baseado em FPGA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Prof. Doutor Arnaldo Silva Rodrigues de Oliveira e de Prof. Doutor Paulo Bacelar Reis Pedreiras, Professores Auxiliares Convidados do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Doutor António Manuel de Brito Ferrari Almeida
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Doutor Francisco Manuel Madureira e Castro Vasques de Carvalho
Professor Associado do Departamento de Engenharia Mecânica e Gestão Industrial da Faculdade de Engenharia da Universidade do Porto

Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar Convidado da Universidade de Aveiro (Orientador)

Doutor Paulo Bacelar Reis Pedreiras
Professor Auxiliar Convidado da Universidade de Aveiro (Co-Orientador)

agradecimentos / acknowledgements

Em primeiro lugar, gostaria de agradecer ao Prof. Doutor Arnaldo Oliveira e Prof. Doutor Paulo Pedreiras por se mostrarem sempre disponíveis a discutir ideias, por todo o seu apoio, todas as críticas construtivas e toda a dedicação demonstrada, funcionando como grande factor de motivação para a realização deste trabalho.

De seguida, quero também agradecer aos membros do Laboratório de Sistemas Embutidos do Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA), em particular ao Eng. Rui Santos, ao Eng. Ricardo Marau e ao Prof. Doutor Luís Almeida, sempre dispostos a ajudar.

Um agradecimento especial a todos os professores e colegas com quem lidei nos últimos 5 anos, em particular ao Prof. Doutor Armando Rocha, pela ajuda no âmbito desta dissertação, e a todos aqueles que partilharam comigo o mesmo espaço de trabalho, em particular à ala direita da sala 303, ao Samuel, ao Nelson, ao Domingos, ao Vítor e também ao Nuno. Foi sobretudo com eles que passei os últimos meses com muitas conversas partilhadas e muitas ideias discutidas.

É necessário mostrar também a minha gratidão a todos os meus familiares, em especial aos meus pais, sem os quais não teria levado esta experiência a bom porto, à Rita, pela paciência, amizade e bom senso e finalmente a todos os colegas e amigos.

A todas estas pessoas, os mais sinceros agradecimentos.

Palavras Chave

Redes de Comunicação, Ethernet, Tempo-Real, Sistemas Distribuídos, *Sniffer*, Analisador de redes, Computação Reconfigurável, FPGAs.

Resumo

A crescente utilização de sistemas distribuídos em aplicações de tempo-real tem levado à criação de protocolos de comunicação cada vez mais complexos e sofisticados. Apesar da rede Ethernet não apresentar características de tempo-real, devido às suas vantagens, têm sido desenvolvidos muitos protocolos de comunicação tempo-real baseados em Ethernet. Nesta dissertação é analisada a importância das arquitecturas distribuídas em aplicações de tempo-real, sendo apresentados alguns conceitos relacionados com esta problemática.

Para avaliar o funcionamento lógico e temporal de uma rede de comunicação é necessário utilizar ferramentas, vulgarmente designadas *sniffer*, que permitem observar o tráfego que nelas circula e os respectivos instantes. Apesar de existirem inúmeras ferramentas deste tipo para Ethernet, o seu desenvolvimento foi conduzido pelos requisitos de redes de dados de uso geral. A maior parte das aplicações existentes destina-se a correr num computador convencional, bastando este estar equipado com uma placa de rede vulgar. No entanto, devido à arquitectura dos sistemas computacionais de uso geral, às suas capacidades multiprogramação e à forma como é efectuado o *time-stamping* das mensagens, estas aplicações não satisfazem as necessidades específicas de alguns protocolos tempo-real, nomeadamente no que concerne à resolução e precisão com que se consegue medir os instantes de transmissão e de recepção das mensagens.

Como resposta às limitações das ferramentas existentes, esta dissertação apresenta um *sniffer* capaz de responder às necessidades específicas dos protocolos de tempo-real. Para isso, a recepção, *time-stamping* e tratamento de mensagens são efectuados com recurso a *hardware* dedicado. Neste contexto, tirou-se partido da utilização de dispositivos lógicos programáveis, em particular das FPGAs (*Field-Programmable Gate Arrays*) e da existência de núcleos sintetizáveis de propriedade intelectual, permitindo implementar de forma relativamente simples a camada MAC de Ethernet utilizando o núcleo *Xilinx LogiCORE Tri-Mode Ethernet MAC* disponibilizado pela *Xilinx*. Os restantes módulos implementados na FPGA têm como objectivo efectuar a escrita das mensagens recebidas e de toda a informação associada na memória para posterior leitura e envio via USB. De modo a intercalar o *sniffer* na rede de uma forma o menos intrusiva possível, foi construído um TAP Ethernet, também conhecido por *Y-splitter*, permitindo efectuar a captura das mensagens em ligações *full-duplex*. Desta forma os dados são duplicados para a FPGA sendo efectuado o seu *time-stamping* e posterior envio de toda a informação via USB para um PC. Os dados são recolhidos pelo PC e armazenados num ficheiro compatível com o *Wireshark*, permitindo que a captura seja aberta e analisada com recurso a ferramentas *standard*.

Foram também definidos mecanismos que permitem exportar a informação capturada para ferramentas de cálculo capazes de gerar gráficos e efectuar uma caracterização do tráfego capturado.

Quanto aos resultados obtidos é possível concluir que a ferramenta construída permite efectuar medições temporais rigorosas, com uma resolução de 10ns e um erro máximo de 100ns. Apresenta no entanto algumas limitações ao nível da transferência via USB, podendo a duração da captura ser relativamente limitada. Caso a taxa de transmissão com que a informação é recebida da rede Ethernet seja superior à taxa com que dados são enviados da FPGA para o PC, a capacidade de armazenamento temporário da FPGA atinge os seus limites e a captura é terminada. Por outro lado, as aplicações baseadas em *software* permitem obter resultados com uma resolução temporal de 1μs e uma incerteza na casa dos milisegundos. Para além disso, em determinadas situações, utilizando ferramentas baseadas exclusivamente em *software* podem ser perdidas mensagens sem que o utilizador seja alertado deste acontecimento.

Key Words

Communication Networks, Ethernet, Real-Time, Distributed Systems, Sniffer, Network Analyzer, Time-Stamping, Reconfigurable Computing, FPGAs.

Abstract

The growing use of distributed systems on real-time applications has originated more and more complex and sophisticated communication protocols. Despite Ethernet network does not have real-time characteristics, due to its advantages, a lot of real-time communication protocols based on Ethernet have been developed. This dissertation analyzes the importance of distributed architectures concerning real-time applications and it also describes some related concepts.

In order to evaluate the logical and temporal performance of a communication network, it is necessary to use tools, commonly named as sniffers, which allow the examination of the traffic. Although there are a lot of these tools for Ethernet, its development was carried out by the requisites of general purpose data networks. The majority of existing applications can be used on conventional computers (an ordinary network interface card is enough). However, owing to the architecture of computational systems, their multiprogramming abilities and to the way time-stamping of messages is done, these applications do not satisfy the specific needs of some real-time protocols, namely regarding the resolution and precision to measure the moments of transmission and reception of messages.

To answer to these limitations, this dissertation presents a sniffer which is able to respond to the specific needs of real-time protocols. Therefore, the reception, time-stamping and management of messages are done using dedicated hardware. In this context, programmable logical devices, specifically FPGAs, and synthesizable cores were used, which permitted, in a simple way, the implementation of MAC Ethernet layer by using *Xilinx LogicCORE Tri-Mode Ethernet MAC*. The remaining modules implemented on FPGA aim the writing of received messages and all associated information in memory for later reading and sending via USB. To intercalate a sniffer within the network in the least intrusive way, a TAP Ethernet, also known as Y-splitter, was built, which permitted the capture in full-duplex connections. This way, information is duplicated to FPGA, then, its time-stamping is done and finally, data is sent to a computer via USB. Information is collected and stored in a file compatible with *Wireshark*, which allows its utilization and subsequent analysis by using standard/regular tools. Mechanisms that permit the treatment of gathered information through calculation tools (which can generate plots and helps to characterize the collected data) were also created.

Concerning the obtained results, it is possible to infer that this tool permits meticulous temporal measurements, with a resolution of 10ns and a maximum error of 100ns. On the other hand, some limitations were experienced, particularly regarding transfer via USB, which results in a more limited

capture if the capacity of temporary storage of the FPGA runs out. This happens when the transmission rate from Ethernet network is higher than the transfer rate between FPGA and the personal computer. Still, software-based applications can present outcomes with a temporal resolution of 1μs and an uncertainty of milliseconds. We can also conclude that in certain situations some messages can be lost without the user is being informed about it.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Enquadramento

As redes de comunicação são actualmente utilizadas em inúmeras aplicações e têm como principal objectivo a partilha de informação entre os diferentes nós constituintes de uma rede. Desta forma é possível repartir a execução de uma tarefa complexa em várias tarefas mais simples. Actualmente, as conexões entre computadores são efectuadas na sua maioria recorrendo ao protocolo Ethernet. Alguns exemplos de aplicação são as redes utilizadas em grande parte das empresas, tais como bancos, nas quais circula uma grande quantidade de informação, ou uma simples rede doméstica, na qual vários computadores têm acesso à Internet ou partilham uma impressora. Nestas situações, quando o utilizador efectua um pedido e este não é imediatamente atendido não há qualquer tipo de consequência para o utilizador uma vez que este pode aguardar alguns segundos até que a informação seja disponibilizada.

Ao contrário das redes vulgarmente utilizadas em escritórios, em ambientes industriais, em sistemas militares e em meios transporte é, tipicamente, necessário o cumprimento de requisitos temporais, isto é, os resultados têm de ser logicamente correctos mas também obtidos atempadamente. Estes factores são preponderantes para o correcto funcionamento do sistema. A comunicação entre robôs, por exemplo, ou o controlo de motores, requer um processamento em intervalos de tempo precisos e bem definidos. De uma falha de comunicação podem resultar danos catastróficos não só a nível material mas também humano. Para este tipo de aplicações é necessário utilizar protocolos de comunicação de tempo-real cada vez mais robustos, sendo alguns deles baseados na rede Ethernet.

A validação do correcto funcionamento de uma rede passa pela utilização de um analisador, habitualmente designado *sniffer*. O objectivo da utilização destas ferramentas é capturar as mensagens e os instantes em que estas circulam na rede. Existem inúmeros programas disponíveis para o fim em questão, o mais conhecido e gratuito é o *Ethereal*, que desde 2006 se passou a chamar *Wireshark* [Wir08]. Estes analisadores foram no entanto desenvolvidos para serem usados em redes de dados de uso geral, não reunindo as características necessárias à análise de alguns protocolos de tempo-real.

1.2 Motivação

Uma vez que a utilização de redes Ethernet de tempo-real em áreas como a automação industrial está em grande crescimento, novos protocolos têm sido desenvolvidos. A criação de

protocolos cada vez mais precisos conduz à necessidade de construir instrumentos de medida capazes de responder às precisões alcançáveis. Desta forma, é necessário criar ferramentas para efectuar medições com uma precisão superior à que os protocolos conseguem alcançar.

Os analisadores baseados unicamente em software, como o *Wireshark* e muitos outros são aplicações que correm sobre um sistema operativo. Devido às características de multiprogramação associadas ao sistema operativo, a atribuição do processador às diferentes aplicações é multiplexada no tempo. Isto significa que o registo do instante em que cada mensagem é recebida nem sempre é exacto. Ao ser recebida uma mensagem, se o processador não estiver disponível, esta é armazenada e só mais tarde processada. Pode-se dar o caso de várias mensagens serem armazenadas e processadas consecutivamente, sendo sinalizadas com uma diferença temporal entre elas correspondente ao tempo de processamento de cada mensagem, originando resultados errados. O tempo que decorre desde que a mensagem é recebida até que seja efectivamente sinalizado o seu instante de recepção, é variável pois depende, entre outras coisas, da utilização do processador, da ocorrência de interrupções que despoletam rotinas de serviço a interrupções, de bloqueios no acesso a recursos partilhados, ou até mesmo das características do próprio *hardware* como por exemplo utilização de acesso directo à memória (*Direct Memory Access* - DMA), de memória *cache* e de arquitecturas *pipelined*.

A incerteza temporal associada às ferramentas destinadas a serem utilizadas em computadores de uso geral, que pode ser da ordem dos milisegundos, não é portanto aceitável quando se lida com alguns protocolos de tempo-real. Torna-se necessário encontrar formas de solucionar este problema, permitindo efectuar medições com precisão da ordem do sub-microsegundo.

A crescente utilização de dispositivos lógicos programáveis tem-se revelado uma mais valia na criação de circuitos dedicados à realização de tarefas específicas. Para além disso, a existência de núcleos de propriedade intelectual permite aumentar de forma bastante significativa a produtividade, permitindo uma prototipagem rápida de circuitos e eliminando os elevados custos associados ao fabrico de circuitos integrados específicos (*Application Specific Integrated Circuit* - ASIC). No caso conreto deste trabalho, a utilização de dispositivos lógicos programáveis permite também a integração de vários componentes num único dispositivo, o núcleo que implementa a camada de controlo de acesso ao meio da rede Ethernet, as memórias para armazenamento das mensagens e toda a lógica adicional.

1.3 Objectivos

O objectivo desta dissertação passa pela criação de um *sniffer* para redes Ethernet capaz de responder às necessidades de análise específicas dos protocolos de tempo-real. Como resposta às limitações anteriormente descritas, pretende-se construir um analisador de protocolos baseados na rede Ethernet de alto desempenho que permita determinar com exactidão os instantes de recepção e transmissão de mensagens. Para isso, a captura e o tratamento temporal das mensagens são efectuados o mais próximo possível da linha de transmissão, mais concretamente ao nível da camada de acesso ao meio (*Medium Access Control* - MAC), recorrendo a *hardware* dedicado e sem causar interferência significativa no comportamento temporal e lógico da rede.

Pretende-se, através da utilização de um dispositivo lógico programável, mais concretamente de uma FPGA (*Field Programmable Gate Array*), assinalar os instantes de recepção das mensagens com uma resolução temporal de 10 nanosegundos, o equivalente ao tempo de um bit quando se lida com taxas de transferência de 100 Mbps. Importa ainda que após

a recepção das mensagens estas sejam transferidas para um computador. Neste, pretende-se desenvolver mecanismos de exportação de dados de forma a que as capturas possam ser processadas recorrendo, por exemplo, a ferramentas de cálculo que permitam a geração de gráficos, como o *octave* ou *matlab*, bem como analisadas utilizando ferramentas *standard* como é o caso do *Wireshark*.

Será dado particular enfoque ao suporte do protocolo *Flexible Time-Triggered protocol on Switched Ethernet* (FTT-SE) [PAG02] [MPA06], desenvolvido no Laboratório de Sistemas Embutidos do Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA) e neste contexto, através da criação de uma aplicação específica pretende-se caracterizar o tráfego capturado com base numa análise estatística e de pior caso.

1.4 Estrutura da Dissertação

Para além da introdução, esta dissertação está organizada da seguinte forma:

- Capítulo 2 - **Conceitos Fundamentais** - Neste capítulo é feita a introdução de alguns conceitos que facilitam a compreensão da dissertação. São abordados os sistemas de tempo-real baseados em arquitecturas distribuídas, o funcionamento da rede Ethernet e a utilização de computação reconfigurável e dispositivos lógicos programáveis de elevada capacidade no projecto de circuitos digitais.

- Capítulo 3 - **Trabalho Relacionado** - Este capítulo caracteriza de forma sucinta os protocolos de tempo-real e apresenta o protocolo FTT-SE. Para além disso, expõe as limitações das ferramentas vulgarmente utilizadas para análise de redes de dados de uso geral e explica por que motivo a sua utilização não é adequada em aplicações de tempo-real. Finalmente, são abordados alguns trabalhos realizados a nível académico relacionados com sistemas de tempo-real distribuídos e é efectuado um resumo das ferramentas existentes no mercado.

- Capítulo 4 - **Componentes de Hardware do Sniffer Desenvolvido** - Neste capítulo é apresentado todo o *hardware* constituinte do *sniffer* sendo a sua arquitectura descrita de uma forma detalhada. Apresenta ainda o resultado da sua implementação em FPGA, isto é, as ferramentas utilizadas, a forma como o projecto se encontra organizado e os recursos utilizados da FPGA. Este capítulo termina efectuando uma descrição da forma como o *sniffer* é intercalado na rede de comunicação.

- Capítulo 5 - **Componentes de Software do Sniffer Desenvolvido** - Este capítulo apresenta o *software* necessário para transferir dados da FPGA para o PC. Inclui o *firmware* de programação do controlador USB que se encontra ligado à FPGA bem como o *software* para recepção dos dados do lado do computador. Descreve ainda a forma como os dados podem ser visualizados e exportados para outros formatos, sendo utilizados pelas ferramentas desenvolvidas para tratamento e análise do tráfego capturado.

- Capítulo 6 - **Testes e Análise do Desempenho** - Neste capítulo são apresentados os resultados obtidos nos testes efectuados, sendo feita uma comparação entre o desempenho da ferramenta construída e de um analisador de redes de uso geral, em particular o *Wireshark*. O *sniffer* desenvolvido é caracterizado em termos das precisões temporais

conseguidas e quanto à capacidade de enviar dados para o PC. Para além disso, é apresentado um exemplo prático da importância do *sniffer* construído, sendo feita uma caracterização das mensagens periódicas próprias do protocolo FTT-SE.

- Capítulo 7 - **Conclusões** - Esta dissertação termina apresentando um resumo do trabalho realizado, as conclusões do trabalho desenvolvido e alguns aspectos que podem ser melhorados, adicionados e tidos em conta em desenvolvimentos futuros.

No Apêndice A - **Geração dos Núcleos de Propriedade Intelectual das Memórias** - e no apêndice B - **Geração do Núcleo de Propriedade Intelectual do Controlador de Acesso ao Meio** - é fornecida informação complementar sobre a forma como os núcleos de propriedade intelectual utilizados neste trabalho foram criados. No Apêndice C - **Lista de Acrónimos** - encontra-se a lista dos acrónimos usados nesta dissertação. Finalmente, a **Bibliografia** apresenta uma listagem dos documentos e publicações que serviram como fonte para consulta.

Capítulo 2

Conceitos Fundamentais

2.1 Introdução

A utilização de arquitecturas distribuídas em aplicações de tempo-real está de tal forma difundida que diariamente as pessoas lidam com arquitecturas deste tipo. A título de exemplo, pode referir-se o sistema *X-by-wire* [X-B08] aplicado ao controlo automóvel. *X-by-wire* é um termo genérico que se refere à substituição de sistemas mecânicos e hidráulicos, como a direcção ou os travões, por sistemas eléctricos/electrónicos. No caso de um sistema de travagem *break-by-wire* [HB98], este é implementado com base num microcontrolador, um sensor no pedal do travão e um sensor e um actuador em cada uma das rodas, permitindo melhorar a travagem, reduzindo a distância necessária para imobilizar o veículo. Esta evolução deve-se não só a razões tecnológicas mas também económicas e ambientais, pois a melhoria em termos de desempenho e segurança é acompanhada por uma diminuição no custo do *hardware*. A remoção do sistema hidráulico permite eliminar a utilização de óleo no sistema, bem como reduzir o peso do veículo e consequentemente o seu consumo, contribuindo desta forma para uma melhoria do meio ambiente. O controlo da direcção ou dos travões tem que ser preciso qualquer que seja a velocidade a que o veículo circula. Todos os acontecimentos imprevistos que ocorrem enquanto o carro circula a alta velocidade devem ser tratados a essa velocidade. Se alguém se atravessar na estrada e o condutor accionar o travão este tem de responder imediatamente. Da mesma forma, a comunicação entre os sensores presentes nas rodas não pode falhar, de forma a imobilizar o carro no mínimo espaço possível. Obviamente, se quando o condutor actuar sobre o veículo, o tempo de espera até que este responda for demasiado longo as consequências podem ser fatais. Um sistema em tudo idêntico mas utilizado na indústria aeronáutica é designado *fly-by-wire*. No caso de uma falha num avião as consequências podem ser ainda mais nefastas.

Os automóveis mais recentes vêm também equipados com um sistema de controlo de estabilidade electrónico (*Electronic Stability Program* - ESP) que permite diminuir o número de acidentes por derrapagem em cerca de 80% [Bos08]. Este sistema baseia-se nos sistemas de anti-bloqueio dos travões e de controlo de tracção que garante que as rodas não deslizam ao arrancar. Para um funcionamento correcto é necessário um conjunto de sensores para medir a velocidade rotacional de cada uma das rodas, um sensor que regista a direcção desejada pelo condutor e sensores que detectam o movimento rotacional do veículo em torno do seu eixo vertical. Periodicamente o sistema verifica se existe alguma divergência entre os comandos do condutor e o comportamento do veículo. Caso o comportamento do veículo não corresponda

aos comandos do condutor, a travagem individual das rodas estabiliza o veículo permitindo ao condutor não perder o controlo do mesmo. Para que o sistema permita salvar vidas é necessário que todas as acções ocorram em intervalos de tempo precisos e bem definidos. Se, por exemplo, a informação relativa à velocidade das rodas não chegar atempadamente ao microcomputador este poderá actuar de forma desapropriada sobre os travões podendo produzir um efeito contrário ao desejado, isto é, conduzir à perda de controlo do veículo causando vítimas que poderiam ser evitadas se o automóvel não estivesse equipado com este sistema.

Com base nas descrições anteriores não admira portanto que um automóvel apresente algumas dezenas de microcontroladores que necessitam trocar informações entre si, sendo necessário encontrar formas de disciplinar as comunicações. Os meios de transporte são um exemplo evidente da importância do cumprimento de requisitos temporais, contudo, este tipo de sistemas alarga-se muito para além deste domínio.

Também na indústria é importante que as comunicações sejam disciplinadas e ocorram sem falhas. A utilização de robots numa linha de montagem deve ser precisa, rápida e robusta. No caso de robôs a movimentar cargas próximas de pessoas é importante que estes não percam o controlo, preservando a integridade física das pessoas. Outro exemplo a considerar pode ser o controlo da temperatura de um forno. Se a comunicação entre os sensores de medição da temperatura e os actuadores não ocorrer em instantes bem definidos, a temperatura pode exceder os limites estipulados causando uma explosão. Se em vez de um forno se considerar o controlo de uma central nuclear, os danos causados poderão ser incomparavelmente maiores, basta pensar nas consequências do acidente nuclear de *Chernobyl*.

2.2 Sistemas de Tempo-Real

Os sistemas computacionais de tempo-real caracterizam-se por apresentar uma dependência face ao tempo, isto é, os resultados têm de ser, para além de logicamente correctos, obtidos de forma atempada. Este tipo de sistemas diferencia-se dos restantes pela execução de tarefas dentro de intervalos de tempo finitos e impostos pelo meio envolvente, estando muitas das vezes associados a um processo físico. De acordo com o *Oxford Dictionary of Computing* um sistema de tempo-real é qualquer sistema no qual o instante de tempo em que a saída é produzida é significativo. Isto deve-se geralmente ao facto da entrada do sistema corresponder a qualquer actividade no mundo físico tendo a saída de se relacionar com ela. O atraso entre o instante correspondente à entrada e o correspondente à saída tem de ser suficientemente pequeno para que se consiga atingir uma resposta temporal aceitável.

Na figura 2.1, baseada no livro *Real-Time Systems: Design Principles for Distributed Embedded Applications* [Kop97] encontra-se representado o diagrama de blocos de um sistema de tempo-real computorizado.

2.2.1 Requisitos

Para além da dependência temporal que caracteriza os sistemas de tempo-real, para que estes funcionem correctamente e possam ser utilizados em aplicações de segurança crítica devem também cumprir requisitos funcionais e de dependabilidade.

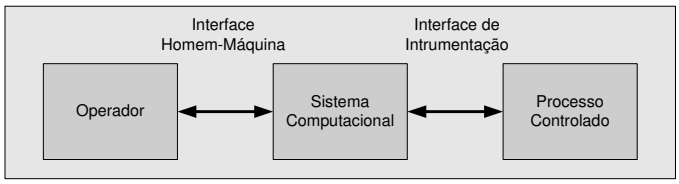


Figura 2.1: Diagrama de blocos de um sistema de tempo-real computorizado (baseada em [Kop97]).

Requisitos Funcionais

Com base na figura 2.1 é possível decompor o sistema de tempo-real nas seguintes acções: aquisição de dados, controlo do processo e interface com o utilizador.

A recolha de dados consiste na obtenção das variáveis de estado do sistema, (entidades de tempo-real), que podem ser contínuas ou discretas. A sua observação gera uma imagem local no sistema controlador que só é válida durante um intervalo limitado de tempo. A validade temporal é imposta pela dinâmica do processo físico a ser controlado e em algumas aplicações, a validade pode ser extremamente curta. Ao conjunto das imagens das entidades de tempo-real dá-se o nome base de dados de tempo-real. Esta deve ser constantemente actualizada de forma a contemplar as mudanças de valor ocorridas nas entidades de tempo-real.

A aquisição de dados pode ser feita recorrendo a duas técnicas de observação:

- A imagem é actualizada periodicamente sendo esta acção despoletada por um sinal proveniente do relógio de tempo-real do sistema (*Time-Triggered*);
- A imagem é actualizada quando ocorre uma mudança de estado na entidade de tempo-real (*Event-Triggered*).

O sistema computacional tem como objectivo actuar sobre o processo controlado para que este atinja um determinado estado dinâmico correspondente à saída desejada. Para isso, o processamento das imagens obtidas através da aquisição de dados deve ser efectuada de acordo com uma estratégia pré-estabelecida e designada algoritmo de controlo. Se este for implementado directamente num computador, o desempenho do sistema é afectado pelas implicações do funcionamento do software e do próprio computador, tais como, a estrutura do código que implementa o algoritmo, o tempo de execução, a influência de outras tarefas, o sistema operativo e o tempo de transmissão de mensagens.

A interface com o utilizador pode ser utilizada como entrada ou saída do sistema. Esta permite ao utilizador manipular o sistema, alterando o seu comportamento, mas serve também para disponibilizar informações relativas ao funcionamento do sistema, tais como relatórios de erros ou um histórico dos valores das imagens de tempo-real.

Requisitos Temporais

Os sistemas de tempo-real apresentam requisitos temporais que advêm das características do ambiente onde se insere o sistema, ou seja, do ritmo de evolução do meio evolvente. Na figura 2.2 estão representadas as restrições normalmente impostas pelos sistemas de tempo-real. Estas restrições estão relacionadas com os atrasos de observação do estado do sistema, com os atrasos de computação dos novos valores de controlo e também da variação dos atrasos anteriores, à qual se dá a designação de *jitter*. De forma a evitar situações que conduzem

à perda de controlo do sistema, as restrições têm que ser cumpridas em todas as instâncias, sendo necessário efectuar uma análise de pior caso e não apenas em termos médios.

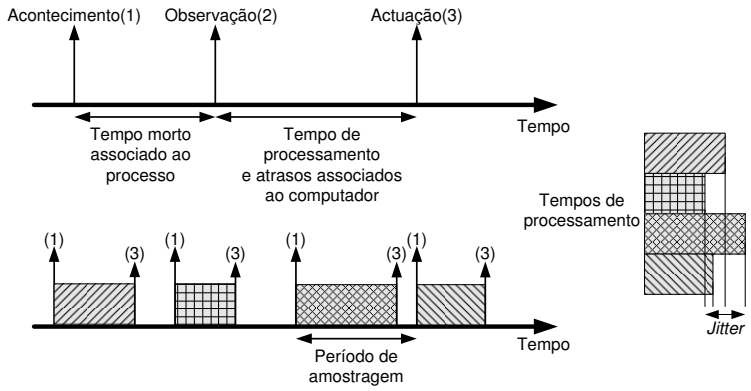


Figura 2.2: Representação das restrições temporais associadas a sistemas de tempo-real.

Os sistemas de tempo-real são habitualmente classificados de acordo com as restrições temporais que lhes são impostas. Se os requisitos temporais não forem ocasionalmente cumpridos e daí não resultarem danos significativos, designam-se sistemas de tempo-real não críticos *(soft real-time)*. É o caso da vídeo-conferência nos quais pequenos atrasos afectam ligeiramente o desempenho do sistema mas a informação continua a ter utilidade mesmo que atrasada.

Por outro lado, se o não cumprimento das restrições temporais provocar uma falha fatal, originando danos catastróficos, os sistemas são considerados de tempo-real crítico *(hard real-time)*. É o caso da leitura de valores provenientes de sensores de temperatura de uma central nuclear que se não forem processados atempadamente podem originar uma explosão.

Requisitos de Dependabilidade

As aplicações de segurança crítica são um exemplo típico da utilização de sistemas de tempo-real resultando daqui requisitos de elevada fiabilidade. Neste tipo de sistemas é importante ter em conta alguns aspectos, tais como: utilização de **interfaces estáveis** entre os subsistemas críticos por forma a evitar a propagação de erros; a existência de **recursos adequados para fazer face aos cenários de pior caso**, oferecendo garantias de qualidade de serviço mesmo em tais cenários; a utilização de **subsistemas autónomos**, cujas propriedades podem ser verificadas independentemente uns dos outros.

2.2.2 Arquitectura

A interligação entre os diferentes componentes de um sistema de tempo-real requer a utilização de redes de comunicação. Estas redes podem ser utilizadas para interligar diferentes aparelhos, é o caso das redes industriais, ou interligando diferentes partes do mesmo aparelho em que os subsistemas têm de ser vistos como um sistema único, é o caso dos sistemas embutidos (*embedded systems*).

Na figura 2.3 estão representadas duas abordagens distintas para criação de um sistema de controlo centralizado. Inicialmente, cada um dos sensores/actuadores era ligado ao computador de processo através de uma ligação ponto-a-ponto. No entanto, de forma a reduzir

consideravelmente a quantidade de cabos necessários para interligação de todos os componentes, esta abordagem evoluiu para uma arquitectura na qual todos os nós constituintes da rede partilham o mesmo barramento. Em ambas as situações existe apenas um único computador de processo e portanto são passíveis de atingir os limites de desempenho na sua unidade central.

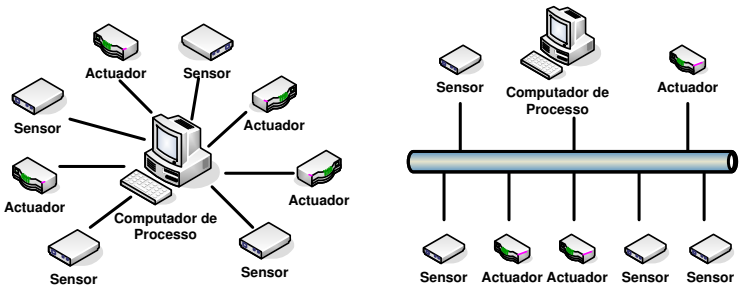


Figura 2.3: Sistemas distribuídos com controlo centralizado.

Uma solução para este problema e que vai de encontro aos requisitos das aplicações de tempo-real passa pela utilização de arquitecturas distribuídas (figura 2.4), tornando o sistema escalável.

Grande parte das aplicações de tempo-real baseia-se em arquitecturas distribuídas. Nestas, existe um conjunto de computadores (nós) a funcionar de forma paralela e descentralizada, conectados através de uma rede, cujo objectivo é concluir uma tarefa comum. Esta abordagem tem sido cada vez mais utilizada devido ao aumento de complexidade dos sistemas e à necessidade de optimizar recursos, cabendo a cada nó a execução de uma sub-tarefa específica e independente das restantes.

Ao contrário de uma arquitectura centralizada, uma arquitectura distribuída deve apresentar menos obstáculos ao crescimento do sistema. Caso haja necessidade de suprir as necessidades de processamento e desde que haja capacidade de comunicação podem ser acrescentados novos nós. Por outro lado, esgotada a capacidade de comunicação pode ser introduzido um *gateway* com o objectivo de interligar redes ou fazer um prolongamento do sistema. Esta característica das arquitecturas distribuídas é habitualmente designada escalabilidade. Outra característica importante destas arquitecturas é a sua dependabilidade, isto é, a capacidade que o sistema tem de continuar em funcionamento com características de tempo-real mesmo na presença de falhas ou erros. Isto é conseguido introduzindo redundância/tolerância a falhas e evitando que os erros se propaguem ao resto do sistema.

Uma vez que todos os computadores podem partilhar os recursos do sistema, hardware, software e dados, é necessário que a informação entre eles seja trocada em instantes perfeitamente definidos. Para isso são utilizados protocolos de comunicação de tempo-real. No caso de sistemas de controlo os vários sensores podem apresentar períodos de amostragem distintos e é possível que as mensagens interfiram umas com as outras afectando os períodos de transmissão e provocando atrasos não controláveis (*jitter*), sendo necessário arranjar formas de minimizar estes efeitos.

2.2.3 Escalonamento

Quando o meio de transmissão é partilhado e o acesso a este tem de ser disciplinado, utilizam-se regras para definir qual a mensagem a ser transmitida num determinado instante,

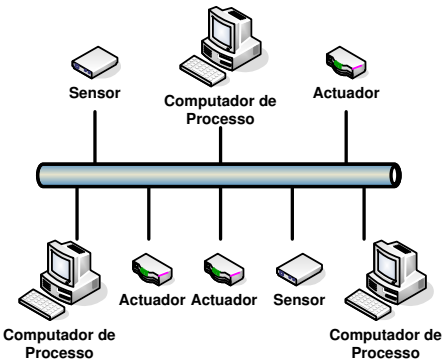


Figura 2.4: Sistema com controlo distribuído.

sendo efectuado um escalonamento das mensagens. Estas podem ser classificadas quanto à sua periodicidade como periódicas, esporádicas ou aperiódicas, encontrando-se representadas na figura 2.5.

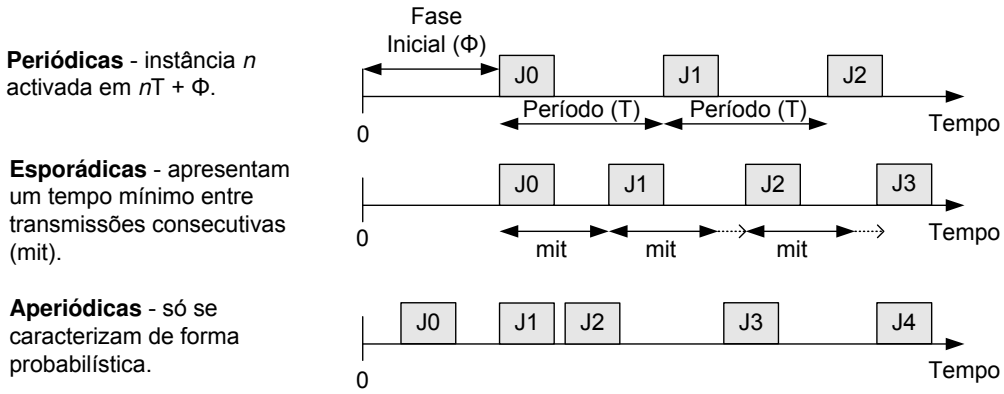


Figura 2.5: Caracterização das mensagens quanto à sua periodicidade.

Dado um conjunto de mensagens e todas as restrições temporais que lhe estão associadas (período, fase inicial, instante máximo para recepção da mensagem - *deadline*) é necessário encontrar uma forma de lhes atribuir o barramento de modo a que as restrições sejam cumpridas. O escalonamento tem como principal objectivo garantir e minimizar a latência na transmissão das mensagens, distribuir o tráfego ao longo do tempo e melhorar os parâmetros das mensagens periódicas, reduzindo por exemplo o *jitter*.

A atribuição de prioridades às diferentes mensagens pode ser feita antes do sistema entrar em funcionamento (*off-line*), não sendo passível de ser alterada e neste caso está-se perante escalonamento estático. Pelo contrário, o escalonamento dinâmico é feito durante o funcionamento do sistema (*on-line*) e sempre que o barramento está disponível decide-se de entre as mensagens prontas qual a próxima a ser transmitida. O escalonamento estático é também passível de ser efectuado *on-line*.

É possível efectuar uma analogia entre o escalonamento de tarefas nos sistemas operativos de tempo-real e o escalonamento de mensagens numa rede de comunicação pois em ambas as situações existe um recurso a ser partilhado. Contudo, no caso dos sistemas operativos pode ser possível interromper a execução de uma tarefa para que o processador seja atribuído

a uma de maior prioridade e diz-se que o sistema é *preemptivo*. No caso da transmissão de mensagens num barramento estas habitualmente não podem ser interrompidas. Para além disso, o escalonamento nos sistemas operativos de tempo-real é tipicamente centralizado pois existe uma lista ordenada com as tarefas prontas para execução. O mesmo não acontece quando se trata de uma rede de comunicação pois é necessário que o escalonador tenha conhecimento do estado das mensagens em todos os nós constituintes do sistema. No caso de uma mensagem originada por um evento, por exemplo, é necessário indicar ao escalonador a prontidão dessa mensagem.

Os principais algoritmos utilizados no escalonamento de tarefas são o *Rate Monotonic* (RM) [But05], no qual é atribuída maior prioridade às tarefas periódicas que apresentam menor período, o *Deadline Monotonic* (DM) [ABRW91], que atribui de forma fixa maior prioridade às tarefas periódicas cuja *deadline* é mais pequena, e o *Earliest Deadline First* (EDF) [But05] que atribui maior prioridade à tarefa cuja *deadline* está mais próxima. Contudo, estes algoritmos centralizados não apresentam o mesmo desempenho quando se trata de uma rede de comunicação devido à sobrecarga das mensagens de controlo (maior *overhead*). Torna-se então necessário encontrar formas mais eficientes e robustas para escalonar as mensagens num barramento partilhado. A título de exemplo pode ser referido o protocolo *Controller Area Network* (CAN) [CIA08] como sendo um protocolo de escalonamento dinâmico. A cada uma das mensagens é atribuído um identificador, definindo este a sua prioridade. A arbitragem é não destrutiva, sendo efectuada ao nível de bit, com base em bit dominante/recessivo. Uma forma de reduzir esta sobrecarga das mensagens de controlo passa também pela utilização de escalonamento centralizado, segundo o qual os nós constituintes da rede tomam decisões baseando-se apenas em informação local. É o caso do protocolo *Ethernet* que no entanto não reúne características que permitam a sua utilização em sistemas de tempo-real crítico, sendo necessário efectuar algumas modificações ao protocolo.

2.3 Ethernet

A Ethernet começou a ser desenvolvida por Robert Metcalfe no centro de investigação da Xerox por volta de 1973, altura em que começavam a ser fabricados os primeiros computadores pessoais. Ele foi encarregue de criar uma rede que permitisse a todos os computadores do centro utilizar uma impressora a laser. Era portanto necessário que a rede fosse suficientemente rápida para enviar os dados para a nova impressora e suficientemente abrangente para interligar todos os computadores. A rede era constituída por um cabo coaxial partilhado por todas as estações e oferecia uma taxa de transmissão de 2.94 Mbps. Em 1976 Robert Metcalfe e David Boggs, seu assistente, publicaram um artigo intitulado *"Ethernet: Distributed Packet-Switching For Local Computer Networks"* [MB76]. Três anos mais tarde Robert Metcalfe abandonou a Xerox e convenceu diferentes fabricantes, entre os quais a Xerox e a Intel, para em conjunto desenvolverem a Ethernet como um protocolo padrão. A sua padronização bem como a sua facilidade de utilização e robustez fazem actualmente da Ethernet o protocolo mais utilizado em redes locais de computadores (*Local Area Network* - LAN).

A Ethernet opera ao nível das camadas física e de ligação de dados do modelo OSI (*Open Systems Interconnection*). Surgiu com uma taxa de transmissão de 10Mbps e permitindo a transmissão de tramas cujo tamanho varia entre 64 e 1518 bytes. Ao longo dos anos esta tem sofrido algumas alterações existindo actualmente diferentes padrões, tais como o *fast* Ethernet e *gigabit* Ethernet permitindo taxas de transmissão até 10Gbps e encontrando-se

em desenvolvimento os padrões 40Gbps e 100Gbps. Neste trabalho será dado particular destaque ao padrão *fast* Ethernet que permite obter taxas de transferência até 100Mbps.

2.3.1 Ligação Física

Inicialmente eram utilizados cabos coaxiais como meio de comunicação, actualmente os mais vulgares e também utilizados neste trabalho são cabos entrançados com conectores modulares 8P8C, conhecidos também como RJ45. Estes são constituídos por 4 pares entrançados e suportam velocidades de 10Mbps, 100Mbps e 1000Mbps, contudo, quanto maior for a taxa de transmissão suportada, maior terá de ser a qualidade dos cabos utilizados. No caso dos cabos UTPcat5, isto é, sem blindagem, e para a taxa de transmissão de 100Mbps o seu comprimento máximo é de 100 metros. Também é possível recorrer à utilização de fibra óptica, no entanto, devido ao seu elevado custo esta é utilizada apenas para ligações envolvendo grandes distâncias.

Para ligação de um dispositivo Ethernet ao canal de comunicação é utilizado um circuito, vulgarmente designado Phy, responsável pela implementação da camada física do modelo OSI, isto é, codifica/descodifica os dados transmitidos sobre os pares entrançados do cabo. Existe também a camada de controlo de acesso ao meio (*Medium Access Control* - MAC). Esta é uma sub-camada da camada de ligação de dados do modelo OSI que recebe e processa os dados provenientes do Phy.

De forma a tornar a interligação entre o MAC e o Phy independente do tipo de Phy foi padronizado um barramento de Interface Independente do Meio (*Media Independent Interface* - MII), desta forma, o controlador de rede pode interagir com qualquer Phy usando a mesma interface de hardware. No caso do padrão *fast* Ethernet este barramento transfere dados usando palavras de 4 bits em cada sentido a uma frequência de 25MHz permitindo desta forma obter a taxa de 100Mbps. Entre as duas camadas existe também uma interface designada (*Management Data Input/Output* - MDIO) constituída por um barramento de 2 bits (relógio e dados) que permite aceder através do MAC aos registos internos do Phy. Todos os dados são transferidos sincronamente com o sinal de relógio MDC (*Management Data Clock*) fornecido pelo MAC cuja frequência máxima é 2.5Mhz. Os dados são transmitidos numa linha *tri-state*, sendo a linha actuada pelo MAC nas operações de escrita e pelo Phy nas operações de leitura. As interfaces descritas anteriormente encontram-se representadas na figura 2.6.

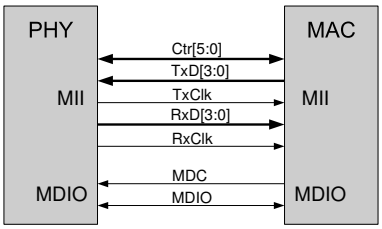


Figura 2.6: Ligação do Phy ao MAC Ethernet.

2.3.2 Algoritmo de Acesso ao Meio

Uma vez que que o meio de transmissão é partilhado por várias estações foi definido um mecanismo de acesso ao meio que assenta num esquema designado CSMA/CD, do inglês *Carrier Sense Multiple Access with Collision Detection*. É um protocolo de partilha do meio

de transmissão que se baseia na detecção da existência de uma transmissão em curso. Uma mensagem para ser transmitida deve esperar que o meio seja detectado disponível e só depois é iniciada a transmissão da mensagem. Contudo, no caso de duas estações detectarem o meio livre e começarem a transmitir em simultâneo irão originar uma colisão de mensagens. A probabilidade de ocorrerem colisões aumenta com a distância a que as estações se encontram umas das outras devido ao atraso de propagação dos sinais eléctricos. Por este motivo, é necessário que os nós escutem a rede enquanto emitem dados. Desta forma, quando uma colisão é detectada as estações param de transmitir e enviam um sinal designado *jam*. Este é usado para notificar todas as estações da ocorrência de uma colisão. Após a colisão as diferentes estações esperam um intervalo de tempo calculado com base no algoritmo de recuo binário exponencial truncado e só depois voltam a tentar transmitir.

Este método de acesso é altamente não determinístico, não sendo possível prever o tempo que uma dada mensagem vai demorar até ser transmitida. Além disso, à medida que o tráfego na rede aumenta e acima de um dado patamar (aproximadamente 60% da largura de banda), devido ao aumento do número de colisões entre mensagens, a quantidade de dados efectivamente transferidos será mais reduzida (fenómeno conhecido como *thrashing*). As colisões representam uma das maiores entraves à utilização da rede Ethernet em sistemas de tempo-real crítico.

2.3.3 Estrutura da Trama

A informação nas redes Ethernet é trocada através de tramas cuja estrutura se encontra representada na figura 2.7. Antes da trama propriamente dita são transmitidos 8 bytes, os primeiros 7 são designados preâmbulo e são constituídos por uma sequência alternada de '0' e '1', usados para sincronização de bit. O byte seguinte é designado *Start Frame Delimiter* (SFD) e serve para os receptores detectarem o inicio da trama, ou seja para sincronização de byte. Para além disso, existe um intervalo de tempo correspondente a 96 tempos de bit entre duas tramas consecutivas. Por este motivo, a taxa de transmissão efectiva de dados apresenta um valor que é sempre inferior a 100Mbps.

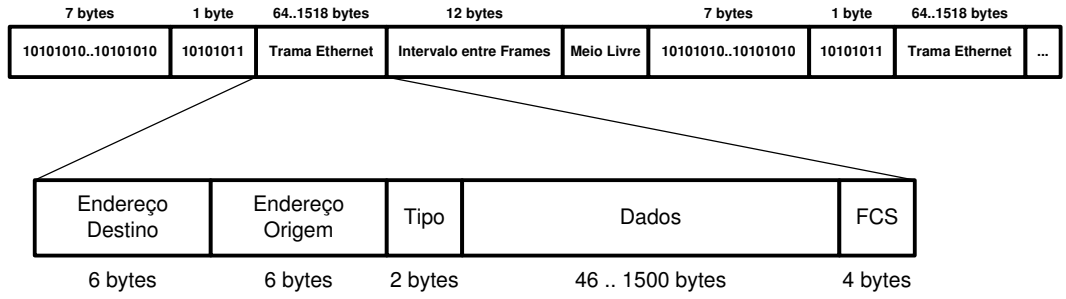


Figura 2.7: Estrutura das tramas Ethernet.

As tramas Ethernet são então compostas por um cabeçalho de 14 bytes. Os primeiros 6 bytes indicam o endereço destino da trama e os 6 bytes seguintes o endereço de origem. Estes endereços são habitualmente designados endereço físico, endereço MAC ou endereço Ethernet. Quanto aos 2 bytes seguintes, se o seu valor for superior a 1500 representa o protocolo ao qual o campo de dados pertence, caso contrário são utilizados para indicar o tamanho do campo

de dados. O tamanho deste campo encontra-se sempre compreendido entre 46 e 1500 bytes. Caso o comprimento da trama seja inferior a 64 bytes, no final deste campo é adicionado um campo designado *pad* por forma a perfazer o comprimento mínimo exigido. Finalmente, surgem 4 bytes (*Frame Check Sequence* - FCS) que permitem detectar erros e cujo valor é calculado com base em todos os campos da trama.

2.3.4 Switched Ethernet

As comunicações numa rede Ethernet ocorriam inicialmente todas no mesmo barramento de comunicação. Desta forma, qualquer informação enviada por um computador era recebida por todos os outros. Para além dos problemas de segurança associados a esta arquitectura, a largura da banda era partilhada por todas as estações, degradando o desempenho da rede. Além disso, uma falha em qualquer ponto da rede traduzia-se na impossibilidade de todos os nós comunicarem. Por este motivo foram criadas topologias em estrela recorrendo à utilização de *hubs*. No entanto, estes replicam a informação recebida numa porta para todas as outras e não eliminam a ocorrência de colisões, mantendo-se o mecanismo de acesso ao meio.

A utilização de *switches* em detrimento de *hubs* permite criar um único domínio de colisão em cada uma das portas do switch (figura 2.8). Estes, analisando as mensagens recebidas, mantêm uma base de dados dos equipamentos de rede ligados a cada uma das portas. Desta forma, as mensagens recebidas são encaminhadas directamente para a porta correspondente ao endereço de destino (*forwarding*). Apenas se o endereço destino não estiver presente na base de dados, as mensagens são encaminhadas para todas as portas do *switch* (*flooding*).

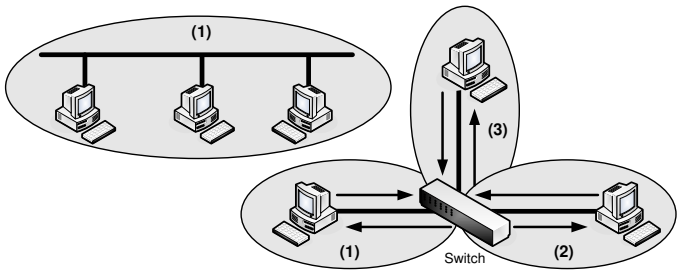


Figura 2.8: Rede Ethernet comutada com ligações *full-duplex*.

Desta forma, se cada um dos nós comunicar com o *switch* e não directamente com outros nós é possível tornar as comunicações *full-duplex*, ou seja, cada uma das estações pode receber e transmitir dados simultaneamente. Uma vez que os cabos utilizam condutores independentes para enviar e receber dados, é possível que a comunicação se faça em ambos os sentidos em simultâneo (figura 2.8). Nestas situações o meio torna-se livre de colisões e o mecanismo de acesso ao meio, que é o principal entrave à utilização da Ethernet como protocolo de tempo-real devido ao seu não determinismo, é abolido. A utilização de *switches* melhora significativamente o desempenho da rede pois subdivide o sistema em várias sub-redes que podem funcionar de forma independente. Contudo, se o ritmo de transmissão de entrada das mensagens for superior ao ritmo de transmissão de saída e a capacidade de armazenamento dos *switches* for excedida, parte das mensagens serão perdidas. Por este motivo, é necessária uma coordenação superior para disciplinar a transmissão de mensagens.

2.4 Dispositivos Lógicos Programáveis

Muitas das ligações Ethernet existentes destinam-se a ser utilizadas em computadores pessoais. Estes são o exemplo mais conhecido de um sistema computacional de uso geral, possuem uma arquitectura que suporta um conjunto de instruções fixo mas que permite a sua utilização em inúmeras aplicações. Uma vez que existem muitas ferramentas que permitem a sua programação, esta torna-se bastante simples e por este motivo, a principal vantagem destas arquitecturas é a sua flexibilidade. Estas não estão no entanto optimizadas para a realização de uma tarefa específica, permitindo obter, em média, um bom desempenho para uma gama alargada de aplicações. Por vezes, é necessário recorrer a sistemas computacionais de elevado desempenho, dado que a capacidade dos sistemas de uso geral não é suficiente para satisfazer todos os requisitos de uma aplicação em particular.

Uma solução que permite aumentar o desempenho de um sistema computacional passa pela criação de circuitos específicos para uma dada aplicação. Desta forma, os sistemas são optimizados executando apenas um conjunto bastante restrito de tarefas. Estes sistemas computacionais especializados não são muitas das vezes programáveis pelo utilizador. É possível projectar um circuito integrado específico (ASIC) optimizado para uma tarefa em particular, conseguindo-se desempenhos bastante elevados e com menos recursos de hardware. No entanto, os elevados custos de projecto e implementação, o risco associado e o tempo de desenvolvimento, apenas tornam a utilização de ASICs uma solução viável no caso de produções em grandes quantidades. Para além disso, a utilização de circuitos integrados específicos não apresenta qualquer flexibilidade dado que não é possível alterar a sua funcionalidade após o seu fabrico. Desta forma, podem-se tornar obsoletos passado pouco tempo.

Os dispositivos lógicos programáveis (*Programmable Logic Device* - PLD), desenvolvidos nas últimas décadas, juntam o melhor das duas técnicas descritas anteriormente: eliminam as desvantagens associadas aos computadores de uso geral, isto é, a não optimização para determinadas aplicações e as principais desvantagens dos ASICs, tais como, custo elevado, tempo de desenvolvimento longo e inflexibilidade. Desta forma, permitem ao utilizador projectar os seus próprios circuitos de elevada capacidade, surgindo neste contexto a computação reconfigurável. Estes apresentam no entanto algumas desvantagens, que têm sido atenuadas, quando comparadas com os ASICs, tais como, maiores consumos de potência, frequências de operação inferiores e menores densidades lógicas.

Os agregados de células lógicas programáveis (FPGAs) começaram a estar disponíveis comercialmente por volta de 1985, tendo sido criadas pela Xilinx. Actualmente, a Xilinx [Xil08d] e a Altera [Alt08] são os maiores fabricantes de FPGAs e líderes do mercado. As FPGAs são exemplo de dispositivos de hardware utilizados em computação reconfigurável. Para além de serem os dispositivos lógicos programáveis de maior capacidade são também os dispositivos utilizados neste trabalho.

2.4.1 Arquitectura Interna das FPGAs

A arquitectura interna de uma FPGA encontra-se representada na figura 2.9. Nela é possível verificar que uma FPGA é constituída por um agregado de blocos lógicos conectados com canais de interligação e cercado por um conjunto de blocos de entrada/saída. Todos estes elementos são programáveis: os blocos lógicos permitem não só a implementação de circuitos combinatórios mas também de circuitos sequenciais, os canais de interligação incluem pistas de ligação pré-fabricadas e interruptores programáveis que permitem a conexão de diferentes

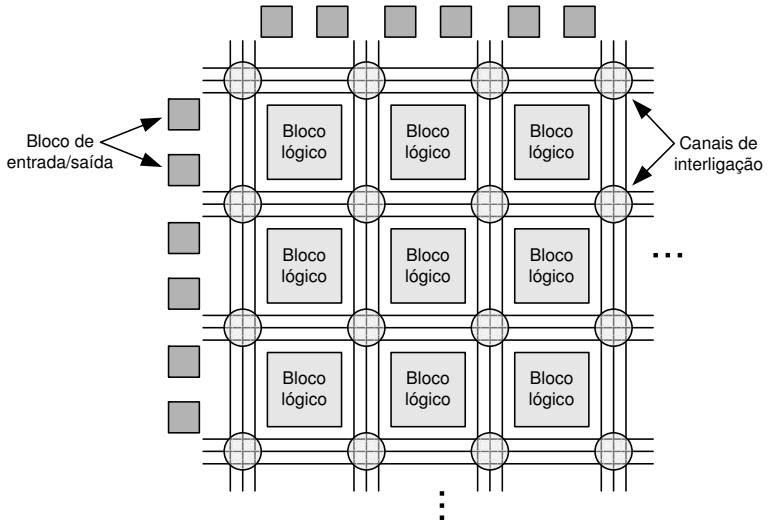


Figura 2.9: Arquitectura interna de uma FPGA.

blocos lógicos. Finalmente, os blocos de entrada/saída fazem a ligação aos pinos da FPGA permitindo a sua ligação ao exterior. Convém referir que o atraso de propagação através dos canais de interligação é um factor importante no desempenho de uma FPGA, sendo este fortemente dependente da forma como a lógica é distribuída e consequentemente da forma como é feita a interligação entre os blocos lógicos. A eficácia das ferramentas CAD (*Computer Aided Design*) reflecte-se portanto bastante no desempenho do circuito.

A implementação de funções lógicas nos blocos lógicos pode ser efectuada de várias formas, no entanto, os principais fabricantes utilizam blocos de memória designados *Look-Up Table* (LUT). Estas memórias permitem armazenar um valor lógico em cada célula. Habitualmente cada LUT possui quatro a seis entradas, permitindo endereçar 16 ou 64 células de armazenamento. Estas células são voláteis, logo sempre que o dispositivo é desligado da alimentação perde a sua configuração. Por este motivo é comum utilizar-se uma memória não volátil para carregar automaticamente as células de armazenamento sempre que a FPGA é ligada.

É comum as FPGAs mais recentes apresentarem também algumas estruturas como por exemplo blocos de memória de maior dimensão, o que possibilita incorporar sistemas completos num único encapsulamento. Em [Dae05] é possível encontrar um resumo de várias características das diferentes famílias de dispositivos da Xilinx. Neste trabalho foram usadas FPGAs da família Spartan-3 por apresentarem uma arquitectura adequada e baixo custo.

2.4.2 Projecto de Circuitos

O projecto de um circuito em FPGA pode ser decomposto essencialmente em três etapas: especificação, síntese/implementação e verificação. Inicialmente é necessário especificar o circuito a ser implementado. Isto pode ser feito recorrendo a ferramentas gráficas que permitem criar um diagrama esquemático do circuito, ou com recurso a linguagens de descrição de hardware. Os blocos básicos devem ser descritos recorrendo a linguagens de descrição de hardware, no entanto, de forma a aumentar a produtividade existem já bibliotecas para cada família de FPGA que incluem blocos básicos que podem ser usados. Existem também ferramentas que permitem gerar componentes parametrizáveis, é o caso do *CORE Generator*

da Xilinx através do qual foi gerado o *Tri-Mode Ethernet Media Access Controller* (TEMAC) [Xil08c] e as memórias utilizadas neste trabalho.

Segue-se a fase de síntese, na qual a especificação é traduzida para os recursos básicos (primitivas) constituintes da FPGA. Após a síntese segue-se a fase de implementação, sendo os elementos do projecto mapeados nos blocos lógicos da FPGA. É atribuída uma posição na FPGA a cada um dos blocos e são feitas as ligações entre os vários componentes. Os algoritmos utilizados asseguram que os blocos que requerem uma comunicação mais intensiva ficam próximos de forma a minimizar o número de segmentos de pistas de ligação utilizados.

A verificação pode ser efectuada de três formas distintas: aquando da especificação é possível efectuar uma simulação funcional que permite validar o comportamento lógico do circuito. Após o mapeamento de todo o circuito na FPGA é possível analisar o comportamento do circuito com base numa simulação temporal na qual são incluídos todos os atrasos lógicos e de encaminhamento. Finalmente, depois da FPGA ter sido configurada é possível verificar a implementação física do circuito. Isso pode ser feito encaminhando os sinais que se pretende observar para pinos da FPGA para que sejam verificados recorrendo a um analisador lógico. Alternativamente, é possível gerar o núcleo de um analisador lógico interno que utiliza os blocos de memória existentes na FPGA para armazenar o valor dos sinais a observar. Estes são posteriormente transferidos para o computador, através de uma interface JTAG (*Joint Test Action Group*) e não ocupando pinos adicionais da FPGA. É disto exemplo o *ChipScope Pro* [Xil08a] da Xilinx.

As FPGAs são portanto uma excelente alternativa à utilização de ASICs devido à sua versatilidade, capacidade e custo reduzido. Por outro lado, ao contrário dos microcontroladores e computadores de uso geral, que apenas suportam um conjunto de instruções implementadas pelo fabricante, estas permitem construir *hardware* específico e dedicado à execução de uma tarefa em concreto.

Capítulo 3

Trabalho Relacionado

3.1 Introdução

A crescente utilização da rede Ethernet em sistemas distribuídos e associados à automação industrial tem conduzido à criação de novos protocolos de tempo-real. Devido à elevada precisão temporal associada a estes protocolos, conseguindo-se obter desempenhos com um *jitter* na casa do microsegundo, torna-se necessário criar ferramentas com capacidade de analisar correctamente o funcionamento e desempenho destes protocolos. Neste capítulo são apresentados alguns protocolos de tempo-real baseados na rede Ethernet e em particular o protocolo FTT-SE.

É feito um levantamento dos trabalhos existentes relacionados com a captura das mensagens com recurso a *software* e a *hardware* dedicado. São também abordadas algumas ferramentas existentes no mercado para análise do tráfego que circula nas redes de comunicação.

3.2 Protocolos de Tempo-Real Baseados em Ethernet

Os protocolos de tempo-real têm vindo a ser desenvolvidos com o objectivo de encontrar protocolos de comunicação capazes de oferecer garantias em termos de determinismo, largura de banda atribuída e latência. Apesar da Ethernet, devido as características apresentadas no capítulo anterior, não preencher os requisitos específicos dos protocolo de tempo-real, segundo Decotignie [Dec01] existem fortes argumentos a favor da sua utilização, tais como:

- custo reduzido devido à produção em grandes quantidades;
- fácil integração com Internet permitindo a utilização de protocolos FTP, HTTP e outros;
- aumento da velocidade de transmissão (tem aumentado constantemente e espera-se que tal continue a acontecer);
- largura de banda suficiente para suporte de desenvolvimentos recentes, tais como multimédia;
- existência de uma grande disponibilidade de equipamentos;
- é amplamente conhecida, suportada e compreendida.

Pelos motivos mencionados, várias abordagens têm sido seguidas com o objectivo de utilizar a rede Ethernet como protocolo de tempo real. Algumas destas abordagens são apresentadas em [PA05].

Da mesma forma que os sistemas de tempo real podem ser classificados de acordo com o tipo de restrições temporais associadas, também os protocolos de tempo real podem ser caracterizados de acordo com as garantias temporais oferecidas. Desta forma, é possível dividir os protocolos de tempo-real em dois grandes grupos.

Num primeiro grupo encontram-se aqueles que não garantem que as mensagens sejam correctamente transmitidas dentro de um intervalo de tempo exacto mas sim dentro de um intervalo de tempo médio. Alguns destes protocolos conseguem atingir desempenhos de tempo-real impondo apenas uma forte limitação na largura de banda utilizada. Neste contexto, grande parte das mensagens são curtas e a utilização do meio é moderada, logo as perdas são reduzidas e na maioria das vezes os requisitos temporais são satisfeitos. Contudo, a largura de banda utilizada por estes protocolos é uma pequena parte da largura de banda disponível, existindo um grande desperdício de recursos. É disso exemplo o protocolo *Network Data Delivered Service* (NDDS) [PCSH99].

Outros protocolos melhoram o comportamento temporal da rede alterando o mecanismo de acesso ao meio. Estes determinam quando devem transmitir com base no estado actual da linha e em informação local. Quando é detectada uma colisão modificam o mecanismo de recuo e retransmissão. Desta forma, tentam diminuir a ocorrência de colisões e tornar a sua resolução determinística, o que não é conseguido com base no algoritmo de recuo binário exponencial truncado. Estes protocolos não eliminam no entanto a existência de colisões, não dando garantias temporais. Apenas permitem efectuar uma caracterização temporal em termos probabilísticos, não sendo adequados para aplicações de tempo-real crítico. É o caso do protocolo *Virtual Time CSMA* [MK85].

Num segundo grupo encontram-se os protocolos que garantem que a transmissão de uma mensagem é efectuada sempre dentro de um intervalo de tempo bem definido.

Uma forma de eliminar as colisões consiste na utilização de um protocolo do tipo *token passing*. Nestes protocolos existe um único *token* que é trocado ciclicamente entre os vários nós e apenas o nó que estiver na posse deste tem direito a enviar mensagens. A ordem pela qual o sinal percorre todos os nós varia consoante o protocolo em questão e limitando o tempo que cada nó da rede permanece na posse do *token* consegue-se obter um comportamento de tempo-real. A título de exemplo pode ser referido o protocolo RETHER [VC94].

Outra solução passa pela atribuição de faixas temporais bem definidas a cada estação, desta forma, todas as estações têm acesso ao meio em instantes de tempo disjuntos. É no entanto necessário que todos os nós estejam sincronizados e tenham a mesma referência temporal. Estes protocolos são designados *Time Division Multiple Access* (TDMA) e são largamente utilizados em aplicações de segurança crítica. Apresentam no entanto um baixo desempenho devido à ocorrência de situações em que é atribuído tempo a estações que não têm dados para transmitir. O protocolo MARS (*MAintenable Real-time System*) [KDK$^+$89] é um exemplo deste tipo de protocolos.

Os protocolos baseados em arquitecturas *Master/Slave* são um dos métodos mais simples para implementar comunicações de tempo-real sobre barramentos distribuídos. Neste tipo de protocolos um dos nós, designado *Master* é responsável por controlar o acesso ao meio de todos os outros, designados *Slaves*. A quantidade de mensagens de controlo enviadas pelo *Master* e o tempo de espera entre mensagens consecutivas contribuem para um baixo desempenho

deste tipo de protocolos. Tanto o protocolo Ethernet *Powerlink* [Pro08b] como o protocolo FTT-SE, apresentado adiante, são exemplos de protocolos baseados em arquitecturas do tipo *Master/Slave*.

A evolução para redes Ethernet comutadas e a criação de diferentes domínios livres de colisões permitiu a utilização de protocolos de comunicação menos rigorosos que os exigidos quando o meio de comunicação é partilhado. No entanto, continua a ser necessário disciplinar as comunicações de forma a resolver problemas de congestionamento nos *switches*. Estes para além de introduzirem *jitter*, apresentam um comportamento não determinístico em termos temporais. Este comportamento encontra-se associado ao algoritmo de encaminhamento das mensagens e ao estado das diferentes filas em que as mensagens são armazenadas. Desta forma, para que se consiga atingir bons desempenhos em termos de *jitter* e latência é necessário um controlo rigoroso de todos os instantes de transmissão, evitando a acumulação de mensagens nas filas do *switch*.

Muitas das técnicas descritas anteriormente são passíveis de ser aplicadas a redes Ethernet comutadas (*Switched Ethernet*). De seguida é apresentado o protocolo ao FTT-SE (*Flexible Time-Triggered - Switched Ethernet*), baseado num mecanismo semelhante ao *Master/Slave*.

3.2.1 Paradigma FTT

O paradigma FTT [FTT07] foi desenvolvido no Laboratório de Sistemas Embutidos do Instituto de Engenharia Electrónica e Telemática de Aveiro. Utiliza um mecanismo designado *Master/Multislave* segundo o qual o *Master* se dirige a vários *Slaves* através de uma única mensagem. Desta forma reduz a carga na rede quando comparado com as técnicas *Master/Slave* convencionais.

O protocolo FTT aplicado a redes Ethernet encontra-se representado na figura 3.1 sendo possível observar que as comunicações estão organizadas em intervalos de tempo com duração fixa designados ciclos elementares (*Elementary Cycle* - EC).

Figura 3.1: Protocolo FTT aplicado a redes Ethernet (baseada em [PGAB05]).

Cada ciclo elementar inicia-se com a transmissão de uma mensagem proveniente do *Master* designada *Trigger Message* (TM) e destinada a todos os *Slaves*. Nesta encontra-se informação relativa às mensagens que devem ser transmitidas no ciclo elementar em questão, bem como a duração de cada uma das mensagens. Os nós têm portanto que estar perfeitamente sincronizados servindo esta mensagem como uma referência temporal comum a todos. O facto da *Trigger Message* ser enviada para todos os *Slaves*, além de diminuir a carga na rede, permite também uma maior eficiência na utilização da largura de banda. Desta forma, todos os *Slaves* descodificam a mensagem em simultâneo, reduzindo os intervalos de tempo sem tráfego na rede. Após a descodificação da TM enviada pelo *Master* todos os *Slaves* aguardam pelo momento exacto em que devem começar a transmitir cada uma das mensagens.

Para além da fase destinada ao envio da TM cada ciclo elementar encontra-se dividido em duas fases. A primeira, logo após a transmissão da TM e descodificação da mesma

por todos os *Slaves*, é destinada à transmissão de tráfego periódico, apresentando todas as mensagens um período múltiplo do período do ciclo elementar. No caso de sistemas críticos é necessário que o período destinado à transmissão de mensagens periódicas seja suficiente para que no pior caso todas as mensagens possam ser transmitidas. Após a transmissão do tráfego periódico segue-se uma janela temporal destinada á transmissão de mensagens assíncronas, por exemplo, mensagens de alarme que são desencadeadas esporadicamente por determinados eventos e mensagens de controlo próprias do protocolo. Devido à divisão temporal entre as duas fases o tráfego periódico não é afectado pelo tráfego aperiódico.

Neste protocolo o escalonamento é efectuado *on-line* e de uma forma centralizada (no *Master*), permitindo desta forma alterar facilmente a política de escalonamento bem como introduzir alterações nos requisitos das comunicações.

A utilização de Ethernet comutada e a possibilidade de estabelecer ligações *full-duplex* permite no entanto que várias estações transmitam em simultâneo. Esta situação encontra-se representada na figura 3.2 baseada no artigo [MPA06] sobre a aplicação do paradigma FTT a redes Ethernet comutadas.

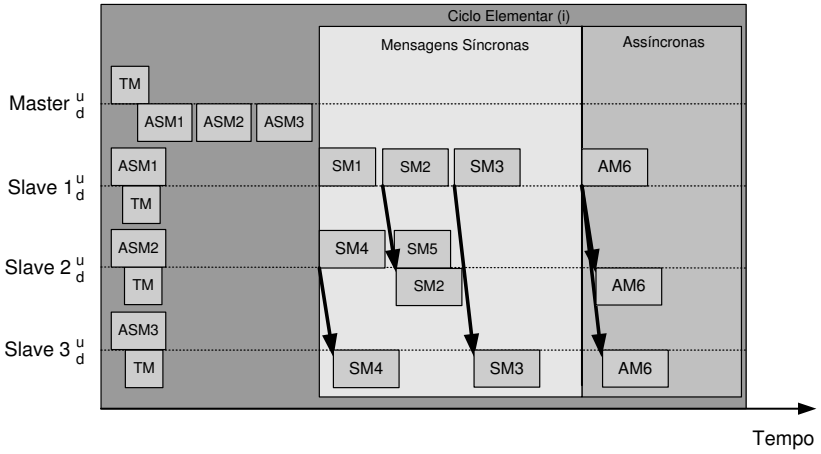


Figura 3.2: Escalonamento baseado no protocolo FTT-SE (baseada em [MPA06]).

Mensagens FTT

As mensagens FTT são encapsuladas em tramas Ethernet, ou seja, são enviadas no campo de dados destas. Todas as mensagens FTT são compostas por um campo de 2 octetos designado *FTT Type* cuja função é identificar o tipo de mensagem, existindo 5 tipos possíveis:

- ***Trigger Message*** - constituída por um cabeçalho de 6 octetos, o qual, para além de bits de controlo, contém o número de mensagens síncronas e assíncronas a ser transmitidas no ciclo elementar em curso. No campo de dados desta mensagem seguem os identificadores das mensagens a ser transmitidas.

- Mensagens de **dados síncronas** e mensagens de **dados assíncronas** - apresentam ambas a mesma estrutura. Estas são constituídas por um cabeçalho de 6 octetos, que contém o identificador da mensagem bem como alguns bits de controlo. O campo de dados é constituído pelos dados a serem transmitidos.

- Mensagens ***Plug and Play*** - utilizadas sempre que um novo nó é acrescentado à rede sendo trocadas entre este e o *Master* para registo do novo nó na rede.

- ***Asynchronous Status Message*** (ASM) - são enviadas dos *Slaves* para o *Master* uma vez por ciclo e uma por nó. Devem seguir exactamente "debaixo" da TM, ou seja, enquanto o *Master* envia a TM, os *Slaves* enviam, em sentido oposto, para o *Master* uma mensagem que contém o estado das várias filas assíncronas existentes em cada um dos *Slaves*. Uma vez que o envio das ASM é sincronizado pela TM anterior e estas são enviadas em simultâneo com a TM, só serão tidas em consideração no ciclo elementar seguinte.

3.3 Ferramentas de Análise de Protocolos

Uma forma de analisar o funcionamento das redes de comunicação passa pela utilização de ferramentas capazes de monitorizar o fluxo de dados que nelas circula. Para além de interceptar o conteúdo das mensagens, de forma a analisar o funcionamento lógico do protocolo, é também necessário determinar os instantes em que cada uma das mensagens é capturada. No caso de sistemas que apresentem uma dependência face ao tempo, a determinação dos instantes em que as mensagens são trocadas é um factor preponderante para avaliar o funcionamento do sistema. Assim sendo, é fundamental determinar os instantes temporais de recepção e de transmissão das mensagens por forma a evitar consequências nefastas que podem advir de um incorrecto funcionamento do sistema. Por este motivo, quando uma mensagem é capturada é lhe atribuído um tempo, correspondente ao instante de recepção, sendo esta marcação temporal habitualmente designada *time-stamping*. A forma como o *time-stamping* é efectuado é extremamente importante quando se lida com aplicações de tempo-real.

3.3.1 Time-Stamping

A determinação rigorosa dos instantes de transmissão e recepção das mensagens em sistemas distribuídos não é apenas importante para averiguar o correcto funcionamento da rede. Em muitas situações, o desempenho do sistema depende da sincronização de diferentes relógios. A norma IEEE 1588, também designada Protocolo Temporal Preciso (*Precision Time Protocol* - PTP) define um protocolo que permite a sincronização com elevada precisão de diferentes relógios em sistemas distribuídos. Este protocolo aplica-se a sistemas de comunicação baseados em redes locais que suportam a troca de mensagens. A exactidão obtida na sincronização dos relógios da rede pode variar entre as dezenas de nanosegundos e algumas centenas de microsegundos, dependendo fortemente do nível em que o *time-stamping* é efectuado (figura 3.3).

Em [WB04] é efectuada uma comparação entre os desempenhos obtidos efectuando o *time-stamping* em *hardware* e *software*. Os resultados demonstram que uma forma de obter resultados precisos consiste na realização do *time-stamping* com recurso a *hardware*. A realização deste em *software* revela-se problemática para muitas aplicações, nomeadamente, de tempo-real. Os resultados podem no entanto ser aceitáveis para aplicações sem elevados requisitos temporais. Neste caso, o melhor que se consegue obter é efectuando o *time-stamping* ao nível do controlador (*driver*) da placa de rede.

As mesmas conclusões podem ser retiradas de [AF05], o qual demonstra a forma como a precisão das medições é afectada pelo *hardware* e *software* utilizado na captura. Para

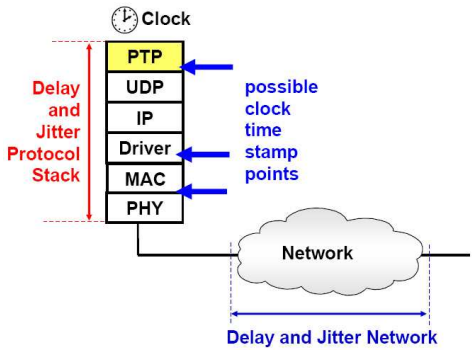


Figura 3.3: IEEE1588: Possíveis pontos para a realização de *time-stamping* (retirado de [WB04].

isso, foram efectuadas várias capturas com base em aplicações a correr num computador pessoal (*Personal Computer* - PC), utilizando uma placa de rede vulgar e utilizando uma placa de captura, designada DAG. Estas placas funcionam de forma independente do sistema operativo fazendo a sua própria marcação do tempo, têm capacidade para armazenar mais mensagens, e permitem ainda a captura de dados específicos de cada pacote. Foram também utilizados diferentes sistemas operativos e PCs com diferentes características (processador e memória). Os resultados demonstram que as precisões dependem fortemente do *hardware* do PC, obtendo-se melhores resultados temporais recorrendo à utilização de *hardware*, neste caso, utilizando a placa DAG. Verifica-se ainda que a utilização de placas de rede vulgares pode conduzir à perda de pacotes sem que estes sejam sequer sinalizados. A utilização destas placas está condicionada a ligações com baixas taxas de transmissão.

3.3.2 Ferramentas Baseadas em Software

Existem inúmeras aplicações que permitem monitorizar o tráfego que circula numa rede, sendo o exemplo mais conhecido o Wireshark [Wir08]. No entanto, existem outros, como por exemplo, o Kismet [Kis08] para redes sem fios e o Snoop [Sno08] para o sistema operativo *Solaris*. Estas aplicações utilizam um PC equipado com uma placa de rede (*Network Interface Card* - NIC) convencional. Esta é colocada num modo de captura em que todos os pacotes que circulam na rede a que a NIC se encontra conectada são recebidos, e não apenas os pacotes a ela endereçados. Isto permite então capturar tudo que entra ou sai num dispositivo de rede instalado num computador. O *Wireshark* é o exemplo mais conhecido desde tipo de aplicações. É no entanto apenas um analisador de protocolos sendo a captura de dados efectuada utilizando a biblioteca *libpcap* ou *winpcap* consoante o sistema operativo em causa. A principal desvantagem associada a este tipo de instrumentos é a sua baixa fiabilidade na determinação dos instantes de recepção das mensagens.

Na figura 3.4 encontra-se representada a arquitectura de um sistema computacional. O sistema de operação é responsável pela criação de uma interface com o hardware, permitindo que este seja operado através das chamadas ao sistema. No entanto, a multiprogramação associada ao sistema operativo, cria uma imagem de aparente simultaneidade na execução de diferentes tarefas pelo mesmo processador. Desta forma, ao efectuar uma captura recorrendo ao *Wireshark*, o processador não se encontra dedicado exclusivamente à realização da captura, existindo concorrência entre vários processos pela posse do CPU. Para além disso,

o barramento PCI (*Peripheral Component Interconnect*) é utilizado para comunicação entre o processador e vários tipos de periféricos, tais como, placas de vídeo, de som, de rede e adaptadores USB. Assim, ao ser recebida uma mensagem na placa de rede, esta necessita solicitar autorização ao árbitro do barramento para poder assumir o controlo deste e transferir os dados para que sejam processados.

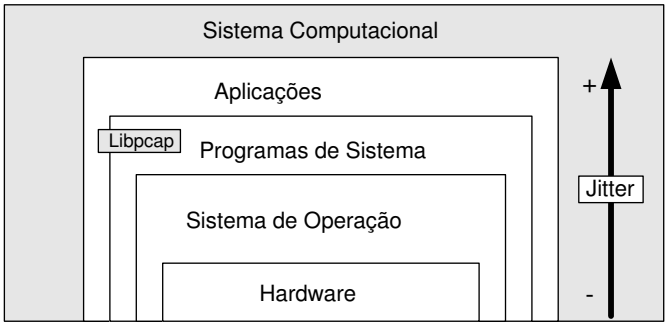


Figura 3.4: Arquitectura de um sistema computacional.

O tempo que decorre desde que uma mensagem é recebida até que seja efectuado o seu *time-stamping* é variável, dependendo de características inerentes ao *hardware* mas também da taxa de utilização do processador e do barramento PCI. A própria placa de rede apresenta alguma capacidade de armazenamento para evitar que caso as mensagens não sejam imediatamente recolhidas, estas não sejam perdidas. Por este motivo, é vulgar várias mensagens serem armazenadas e mais tarde transferidas e processadas consecutivamente, originando resultados errados. A marca temporal que lhes é associada não está relacionada com o instante de recepção mas sim com o instante de processamento. Ao serem processadas consecutivamente duas mensagens, a diferença temporal entre elas será correspondente ao tempo de processamento da primeira e não ao intervalo de tempo decorrido entre a recepção das duas, originando resultados errados.

Estas aplicações não permitem portanto avaliar correctamente o desempenho de uma rede de tempo-real, podendo muitas das vezes conduzir a resultados errados e enganadores. Uma possível solução para este problema passaria pela realização do *time-stamping* na própria placa de rede, desta forma, toda a incerteza inerente ao funcionamento dos computadores seria eliminada e o desempenho da ferramenta seria bastante superior.

3.3.3 Ferramentas Baseadas em Hardware

Pelos motivos já mencionados, efectuar o *time-stamping* com recurso a uma aplicação a correr num PC é uma solução inadequada quando se lida com protocolos de tempo-real que permitem obter precisões da ordem de microsegundos. A solução passa então, por efectuar o *time-stamping* mais próximo da rede de comunicação. Para isso torna-se necessário recorrer a *hardware* dedicado à realização de uma tarefa específica reduzindo desta forma a incerteza associada às medições efectuadas.

Existem no mercado diversas ferramentas com características distintas e que permitem avaliar o desempenho das redes de comunicação.

Em [Yok07b] é apresentada uma ferramenta (AE5501) comercializada pela *Yokogawa* e com um valor próximo de 3000€. Esta é compatível com os padrões Ethernet 10Mbps,

100Mbps e 1Gbps e permite efectuar testes de conectividade na rede, medições de latência, taxas de transmissão e perdas de pacotes. Esta ferramenta tem capacidade para gerar tráfego, podendo ser utilizada em inúmeras aplicações. Permite armazenar resultados de 100 medições efectuadas podendo estes ser posteriormente transferidos para um computador. É possível incluir vários aparelhos na rede de comunicação e configurar os mesmos remotamente utilizando o protocolo *Telnet*. Permite obter uma resolução temporal na medição da latência de 1μs (com um erro máximo de 3μs) para ligações 10Mbps e uma resolução de 100ns (com um erro máximo de 300ns) para ligações de 100Mbps e 1Gbps.

Outra ferramenta semelhante é apresentada em [Anr08a], apresentando um valor comercial da mesma ordem [Eur08] [Con08]. Esta encontra-se equipada com um ecrã táctil, podendo ser utilizada sem qualquer monitor ou teclado externo. Apresenta ainda duas interfaces USB, possibilitando a sua ligação a uma unidade de armazenamento externa ou a uma impressora, sendo possível imprimir relatórios na forma de tabelas e gráficos. Tal como a ferramenta anterior também esta permite exportar os dados para um computador no formato CSV (*Comma Separated Values*). Através da aquisição de *software* adicional é possível controlar até 8 equipamentos remotamente utilizando a rede Ethernet. Para além disso, é possível sincronizar vários dispositivos via GPS e efectuar medições de latências entre equipamentos que se encontram bastante afastados. Como é possível verificar em [Anr08b], esta ferramenta efectua a descodificação de aproximadamente 15 protocolos, permite introduzir erros na rede, originando colisões e gerando mensagens com tamanhos inválidos e permite efectuar contagens de vários acontecimentos, como por exemplo, o número de mensagens recebidas e transmitidas ou número de vezes que ocorreu um determinado erro. Permite ainda efectuar medições de latência com uma resolução máxima de 1μs. Outros equipamentos com características semelhantes podem ser encontrados em [Opt07].

Outras ferramentas foram desenvolvidas para serem utilizadas com o auxílio de um computador pessoal. Em [Yok07a] é apresentado um equipamento também da *Yokogawa*. Este é um modelo mais avançado que o apresentado anteriormente. É capaz de avaliar o desempenho de equipamentos de rede, como *switches* e *routers* comparando o tráfego gerado pela ferramenta com o tráfego capturado. Disponibiliza várias entradas, às quais podem ser ligados diferentes segmentos da rede permitindo detectar situações indesejáveis, tais como, perdas ou duplicação de pacotes, latências anormais, entre outras. Estas informações, bem como o instante de ocorrência, são armazenadas para posterior análise permitindo desta forma efectuar análises durante longos períodos de tempo. Para além disso, algumas versões deste equipamento permitem capturar todo o tráfego que circula na rede de comunicação. Cada uma das portas, às quais é possível ligar os segmentos da rede em análise, é constituída por duas FPGAs, uma para recepção e outra para transmissão de dados. Para além disso, existe uma terceira FPGA responsável pela recolha de dados estatísticos das restantes, comunicando com a unidade de processamento através de um barramento PCI.

Uma ferramenta semelhante, comercializada pela NetScout (S4000), pode ser encontrada em [Net08]. Tal como a anterior disponibiliza várias interfaces, suportando diferentes topologias, tais como Ethernet, *Gigabit* Ethernet, *Token Ring* e ATM (*Asynchronous Transfer Mode*). Estas ferramentas apresentam elevada capacidade de processamento e armazenamento. No caso do modelo S4000 este vem equipado, por exemplo, com um processador *Pentium* 4, 3.0GHz, 2GB de memória RAM e um disco com capacidade para 80GB. O seu valor ronda os 25000€[Nan05], não incluindo o *software* que pode custar até 20000€.

O acesso a estas ferramentas é habitualmente efectuado remotamente via *Web* ou uti-

lizando aplicações próprias, disponíveis tipicamente para o sistema operativo Windows. Estas aplicações permitem não só analisar os resultados mas também configurar as ferramentas antes de estas entrarem em funcionamento. São habitualmente passíveis de ser partilhadas por vários utilizadores e disponibilizam uma porta Ethernet para acesso do utilizador, o que permite isolar a rede a ser analisada da rede de transporte.

O *InfiniStream Network Management* [Man08] apresenta características ligeiramente distintas dos anteriores. Destina-se a capturar toda a informação que circula na rede durante longos períodos de tempo. Suporta até cinco conexões, oferecendo para isso uma capacidade de armazenamento que pode ir de 2TB até 15TB. A informação armazenada pode posteriormente ser analisada recorrendo a um computador utilizando *software* desenvolvido para o efeito. Tal como os anteriores dispõe de uma interface para configuração e inicialização do sistema, mas também de uma consola que disponibiliza vários mecanismos para procurar e examinar o tráfego armazenado. Existem várias versões deste aparelho, variando a capacidade de armazenamento e as topologias de rede suportadas. O seu valor médio ronda os 40000€[APC05].

Para além destes equipamentos destinados a analisar redes Ethernet existem também alguns destinados a outros protocolos de tempo-real, é o caso do *CanAnalyser* utilizado para o protocolo CAN [Can08] e o *ProfiTrace* [PRO08a] para o protocolo ProfiBus. Estas ferramentas funcionam conectadas a um PC, sendo usada uma ligação USB (*Universal Serial Bus*) para transferência de informação para o PC, permitindo desta forma analisar o tráfego capturado.

As ferramentas destinas à análise de redes Ethernet apresentam no entanto como principal desvantagem o seu elevado custo. Por este motivo não são adequadas para uma instalação permanente na rede, permitindo um diagnóstico constante do tráfego que nela circula. Para além disso, algumas delas apresentam uma capacidade de armazenamento de informação reduzido, limitando bastante o intervalo de tempo em que é possível analisar o tráfego.

Em [DFF+06] é apresentado um instrumento de medida alternativo, desenvolvido no meio académico e capaz de avaliar as características de redes Ethernet de tempo-real. Este é constituído por um conjunto de sondas distribuídas pela rede de comunicação sendo cada uma delas utilizada numa ligação *full-duplex*. Cada uma delas é responsável por efectuar o *time-stamping* das mensagens capturadas e o reencaminhamento das mesmas para uma estação de monitorização. O reencaminhamento é efectuado encapsulando a mensagem recebida numa mensagem de maiores dimensões com alguma informação adicional. Esta informação permite saber com exactidão o instante em que a mensagem foi recebida mas também a sonda em que foi recebida. Todas as mensagens recebidas nas diferentes sondas são enviadas para uma única estação de monitorização. Para isso é utilizada uma rede auxiliar paralela à rede em análise, sendo esta também responsável pela sincronização temporal de todas as sondas permitindo comparar o *time-stamp* de mensagens capturadas em sondas distintas. A largura de banda da rede de medição deve ser a mais elevada possível, permitindo enviar todas as mensagens capturadas para a estação de monitorização. Contudo, uma vez que esta não apresenta quaisquer requisitos temporais as mensagens podem ser armazenadas e atrasadas antes de serem enviadas para a estação de monitorização. Esta tem como principal função proceder ao desencapsulamento das mensagens recebidas. Uma vez que a ferramenta se destina a analisar redes Ethernet e *fast* Ethernet, na rede de medições é utilizado o padrão *gigabit* Ethernet ofere-

cendo uma largura de banda dez vezes superior e permitindo tirar partido das frames "jumbo", cujo tamanho pode exceder largamente o tamanho das frames Ethernet convencionais, para encapsular as mensagens capturadas. A estação de monitorização foi adaptada partindo de software já existente, em particular do *wireshark*. Entre este e a biblioteca responsável pela captura (*libpcap*) foi adicionada uma camada de software, permitindo recuperar as mensagens capturadas, os *time-stamps*, e toda a informação adicional. Para além disso foi introduzindo um menu para configuração das sondas permitindo a criação de filtros capazes de seleccionar o tráfego a ser capturado.

Capítulo 4

Componentes de Hardware do Sniffer Desenvolvido

4.1 Introdução

Neste capítulo é efectuada uma descrição de todo o *hardware* constituinte do *sniffer* construído no âmbito deste trabalho. É seguida uma abordagem que parte de uma perspectiva geral de toda a ferramenta para uma descrição detalhada de cada um dos blocos constituintes do sistema justificando as várias opções tomadas. Segue-se uma secção na qual é efectuada uma análise dos recursos utilizados na modelação e implementação do circuito em FPGA. Este capítulo termina descrevendo o circuito que permite intercalar o *sniffer* no segmento de rede a analisar.

4.2 Arquitectura

O *sniffer* foi projectado com a intenção de efectuar medições temporais rigorosas introduzindo o mínimo de perturbação possível na rede de comunicação. O seu diagrama de blocos está representado na figura 4.1. Este deve ser intercalado na ligação em que se pretende efectuar a captura de dados. Para isso foi construído um circuito capaz de duplicar os dados para a FPGA sem interferir significativamente no funcionamento da rede. Este componente é habitualmente designado *TAP* Ethernet e é descrito com mais rigor adiante. A ligação entre a linha de transmissão e o dispositivo lógico programável é efectuada utilizando dois Phys Ethernet, permitindo desta forma efectuar capturas em ligações capazes de operar em modo *full-duplex*. A FPGA é o dispositivo responsável pelo processamento dos dados provenientes do Phy. Esta, para além de registar os instantes em que as mensagens são capturadas, procede ao envio, quer das mensagens quer da informação que lhe está associada (tempo e tamanho), para um computador através de uma ligação USB. A interacção com o utilizador é efectuada usando dois botões que permitem iniciar e terminar a captura sempre que o utilizador assim o entender.

4.3 Estrutura Interna

O funcionamento do *sniffer* pode ser decomposto em duas tarefas distintas. A primeira está relacionada com a aquisição e processamento de dados provenientes da rede Ethernet.

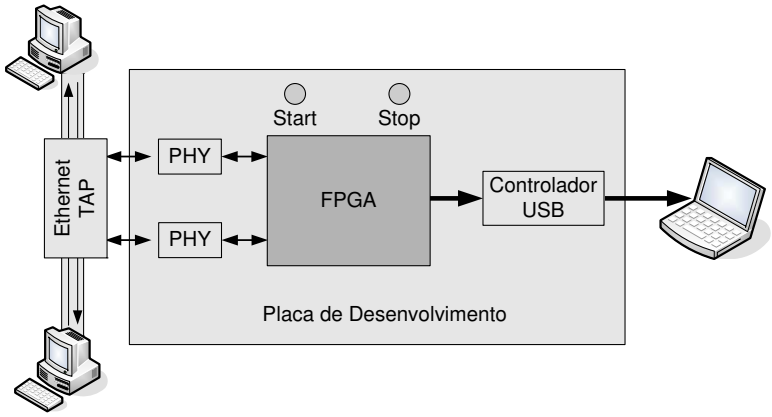


Figura 4.1: Arquitectura do *sniffer* construído.

A segunda tem a ver com o envio de dados via USB para um computador. Uma vez que a ferramenta suporta ligações *full-duplex*, é necessário que esta tenha capacidade para receber e processar em simultâneo duas mensagens completamente independentes. Por outro lado, ambas as mensagens são enviadas para o computador através da mesma ligação USB, não sendo adequado enviar duas mensagens em simultâneo. Isto iria complicar não só o protocolo de comunicação entre a FPGA e o PC como também aumentar o nível de processamento necessário do lado do PC. Para além disso, um protocolo de comunicação mais complexo, iria sacrificar a largura de banda efectivamente atribuída à transmissão de mensagens devido a um maior *overhead* de informação. Por este motivo é necessário que o *sniffer* apresente capacidade para armazenar temporariamente a informação recebida. De forma a simplificar os mecanismos de gestão de acesso à memória optou-se por utilizar memórias independentes para cada um dos sentidos da ligação. Por outro lado, é necessário armazenar não só a mensagem recebida mas também a informação que lhe está associada, tendo-se optado por utilizar também duas memórias independentes: uma para a informação de controlo associada a cada mensagem e outra para o conteúdo da mesma, perfazendo um total de quatro memórias. A utilização de uma memória para dados e outra para controlo permite que a escrita se efectue em ambas em simultâneo. Desta forma, quando uma mensagem começa a ser recebida esta é escrita na memória de dados ao mesmo tempo que o seu *time-stamp* é armazenado na memória de controlo. A figura 4.2 representa a descrição efectuada anteriormente.

Convém também referir que todo o funcionamento da ferramenta é controlado com base em 2 botões de pressão utilizados para iniciar e terminar a captura. Um dos botões serve para fazer *reset* ao sistema e iniciar a captura. Ao ser pressionado, os Phys são configurados e o sistema entra em funcionamento. Se a recepção de uma mensagem se encontrar em curso esta é descartada, começando a captura na mensagem seguinte. O outro botão é utilizado para terminar a captura. Nesta situação, se alguma mensagem estiver a ser recebida, esta é recebida até final e a transmissão via USB termina apenas quando não houver qualquer mensagem armazenada no *sniffer*.

De seguida são apresentados com mais detalhe os principais blocos constituintes do *sniffer*.

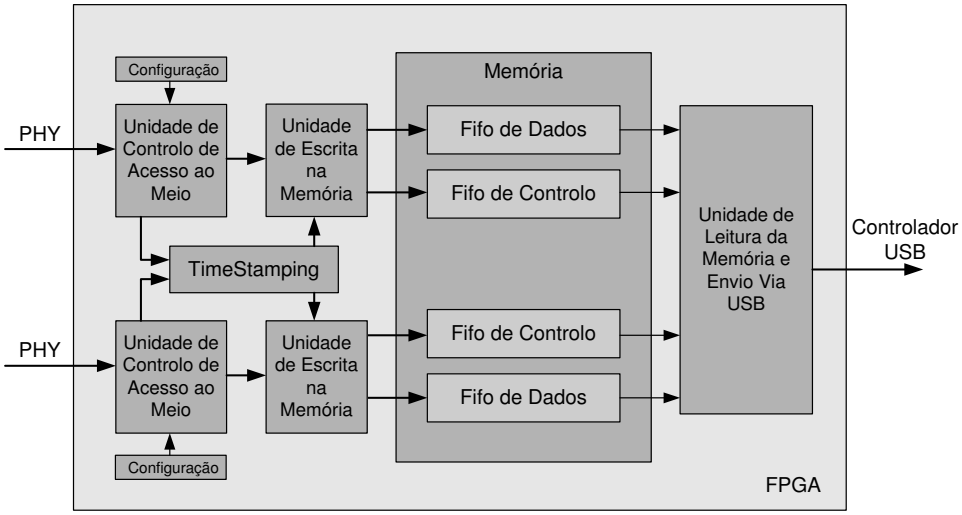


Figura 4.2: Estrutura interna do *sniffer* construído.

4.3.1 Controlo de Acesso ao Meio

Como já foi referido, a interface entre o meio de transmissão e a FPGA efectua-se com recurso a dois Phys Ethernet. Estes por sua vez, comunicam com a FPGA através de um barramento MII, sendo utilizado para processamento dos dados provenientes do Phy, um núcleo de propriedade intelectual da Xilinx (*LogiCORE Tri-Mode Ethernet MAC*) [Xil08c], que implementa a camada MAC do protocolo Ethernet. Em anexo é possível observar a forma como núcleo foi gerado, sendo também justificadas as opções tomadas.

Uma vez que do ponto de vista do *sniffer* apenas é relevante a recepção de tramas, o esquemático do núcleo sintetizado pode ser simplificado e encontra-se representado na figura 4.3. Nesta é possível observar os sinais de interligação do MAC ao Phy Ethernet, uma interface de configuração (*Management Interface*) e uma interface que disponibiliza os sinais utilizados para processamento das mensagens recebidas (Cliente).

Implementação

Em [Xil06] pode ser encontrada mais informação sobre todos os portos bem como informação relativa à configuração do Phy e do MAC, contudo, de seguida é apresentada de forma muito resumida a forma como o TEMAC foi utilizado.

Antes deste e do Phy entrarem em funcionamento, é necessário proceder à sua configuração. Para isso, foi construída uma máquina de estados sensível ao flanco ascendente do sinal *hostclk*, correspondendo cada um dos estados à realização de uma operação de escrita sobre um registo. As alterações efectuadas ao conteúdo dos registos em relação aos seus valores predefinidos tiveram como finalidade: inibir a transmissão de mensagens, activar a recepção e configurar a taxa de transferência para uma velocidade de 100Mbps.

Os sinais disponibilizados do lado do cliente, e que permitem o tratamento das mensagens recebidas, são todos eles síncronos com o flanco ascendente do sinal de relógio *rxgmiimiiclk*, de 25 MHz, obtido a partir do Phy. Contudo, uma vez que o barramento MII transfere dados usando palavras de 4 bits a uma frequência de 25MHz e como os dados são disponibilizados num barramento de 8 bits, estes são actualizados a metade da frequência. Por este motivo

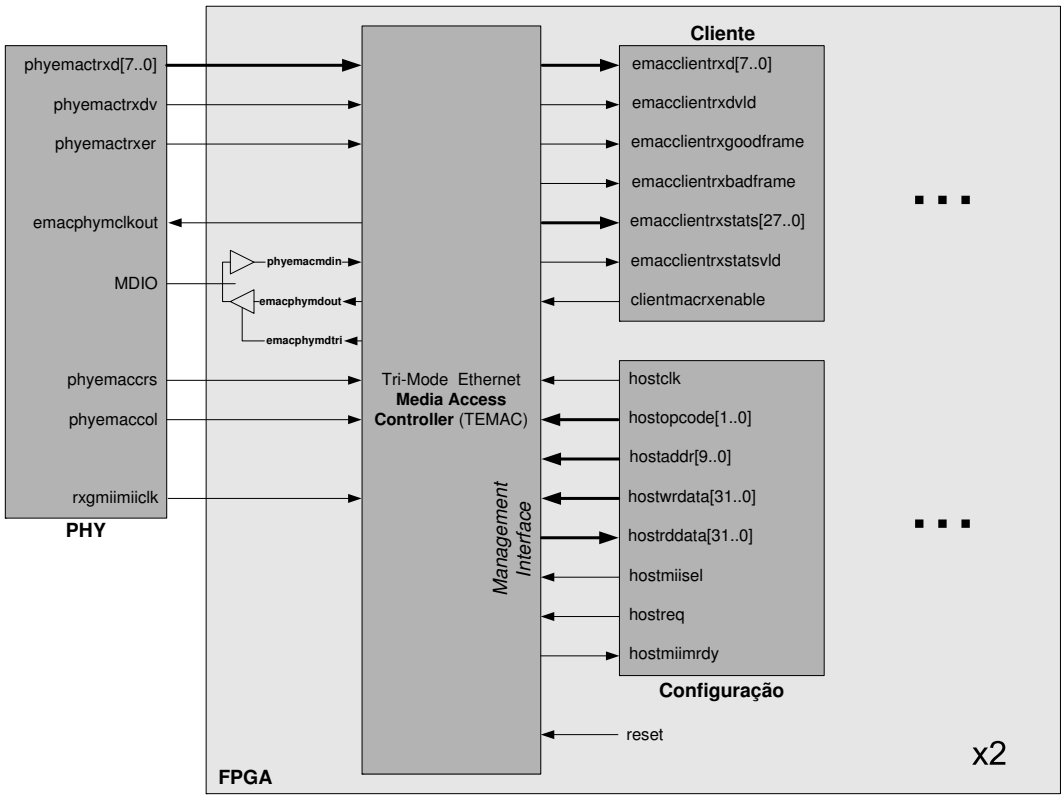


Figura 4.3: Diagrama de blocos de interface da FPGA com o TEMAC.

existe um sinal de 12.5Mhz, designado *clientmacrxenable*.

Para além deste sinal, existem dois barramentos, e um sinal de controlo associado a cada um deles. O sinal de controlo tem como objectivo indicar a validade dos dados presentes no barramento. O sinal *emacclientrxd* é um barramento de 8 bits no qual os dados são disponibilizados, sendo o *emacclientrxdvld* o sinal de controlo associado. O outro barramento, designa-se *emacclientrxstats* e contém informações diversas, entre as quais o tamanho da mensagem que acabou de ser recebida. O sinal de controlo associado é o *emacclientrxstatsvld*. Finalmente, os sinais *emacclientrxgoodframe* e *emacclientrxbadframe* servem para indicar o sucesso na recepção de uma mensagem ou indicar que a trama foi recebida com erros (colisões, FCS incorrecto), respectivamente. Estes dois não são no entanto utilizados pois poderá ser útil ter conhecimento de todo o tráfego que efectivamente circula na rede. Para além disso, o *sniffer* destina-se a ligações *full-duplex* livres de colisões e o mecanismo de detecção de erros se necessário poderá ser implementado em *software* do lado do PC.

Explicada a forma como os dados são adquiridos e disponibilizados pelo TEMAC é então necessário efectuar o seu processamento e armazenamento temporário em memória.

4.3.2 *Time-Stamping*

O módulo designado *TimeStamping*, como o próprio nome indica, é utilizado para efectuar a contagem do tempo e o *time-stamping* das mensagens que são recebidas. Este módulo é comum aos dois "canais" de recepção do *sniffer*, garantindo a mesma base temporal para todas as mensagens, independentemente do sentido em que estas se deslocam. O diagrama

de blocos deste módulo encontra-se representado na figura 4.4.

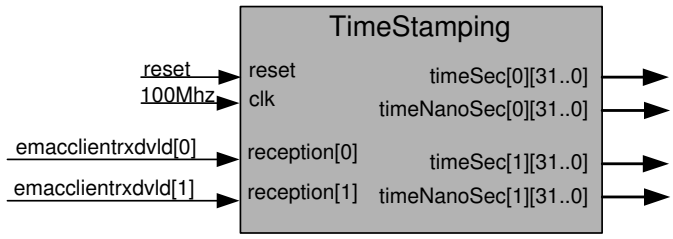


Figura 4.4: Diagrama de blocos do módulo responsável pelo *time-stamping* das mensagens.

Implementação

Para temporização do sistema é utilizado um sinal de relógio de 100Mhz, permitindo uma resolução temporal de 10ns, o equivalente ao tempo de um bit nas ligações *fast* Ethernet. Nos flancos ascendentes do sinal de relógio o valor do contador é incrementado e o valor do sinal *emacclientrxdvld* é analisado. Uma transição de '0' para '1' neste sinal corresponde ao início da recepção de uma mensagem pelo que o valor do contador é armazenado num registo. Este valor mantém-se disponível até recepção da mensagem seguinte sendo entretanto armazenado em memória.

Uma vez que não existe qualquer relação entre o sinal *emacclientrxdvld* e o relógio de 100Mhz, para evitar problemas de sincronismo o sinal *emacclientrxdvld* é amostrado nos flancos descendentes do relógio, garantindo que nos flancos ascendentes se encontra estável. Desta forma obtém-se um erro máximo de 15ns na realização do *time-stamping*. A situação de pior caso encontra-se representada na figura 4.5 e ocorre quando a recepção da mensagem se inicia imediatamente após um flanco descendente do sinal de relógio (1). Assim, a transição no sinal *emacclientrxdvld* só é detectada no flanco descendente 2, sendo o *time-stamping* efectuado no flanco ascendente imediatamente a seguir(3).

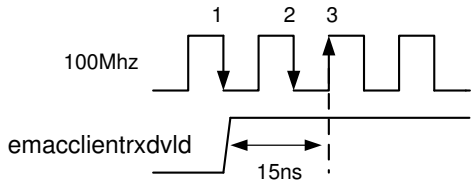


Figura 4.5: Erro máximo associado à realização do *time-stamping* das mensagens.

De seguida é apresentada a forma como esta informação, bem como a informação proveniente do TEMAC é armazenada.

4.3.3 Memórias

Como já foi referido, existem duas memórias distintas associadas a cada um dos sentidos de comunicação. A **memória de dados** é utilizada para armazenar todos os campos constituintes da trama Ethernet. No entanto, caso não exista espaço suficiente para armazenamento da trama completa esta é truncada ao número de bytes disponíveis na memória. Na situação

limite em que a capacidade de armazenamento se esgota não são capturados dados. No entanto, na memória de controlo é armazenada informação relativa à trama cujo conteúdo foi descartado. Esta situação pode ocorrer devido a algumas limitações na largura de banda disponível para envio da informação para o PC e é explicada com mais detalhe no capítulo seguinte. Uma vez que pode ser vantajoso armazenar sempre informação que permita identificar a trama recebida, ao gerar o ficheiro de configuração da FPGA é possível especificar o número máximo de bytes a ser capturados de cada trama. Desta forma, a utilização da memória é repartida de uma forma mais uniforme por todas as mensagens. Poderá não ter interesse capturar na integra o campo de dados de uma mensagem, sendo mais vantajoso ter por exemplo no máximo 100 bytes de cada mensagem.

Quanto à **memória de controlo**, esta é utilizada para registar:

- O instante de recepção da mensagem, sendo utilizados 32 bits para indicar o segundo desde o início da captura mais 32 bits para indicação do nanosegundo dentro do segundo em questão;
- O tamanho total da mensagem recebida, sendo utilizados 16 bits;
- O número de bytes capturados da mensagem (16 bits). Este valor pode ser diferente do tamanho total da trama em duas situações: quando a capacidade da memória de dados atinge o seu limite ou quando o utilizador limita a quantidade máxima de dados a ser capturados.

Por vezes verifica-se que a taxa com que as mensagens são recebidas é superior à taxa de transmissão com que estas são enviadas para o PC. Por este motivo, e para evitar que uma possível perda de mensagens passe despercebida ao utilizador, quando esta memória atinge a sua capacidade máxima de armazenamento a captura em curso é terminada.

Implementação

A escolha das memórias utilizadas recaiu sobre memórias do tipo FIFO (*First In First Out*), criadas utilizando o gerador de núcleos de propriedade intelectual da *Xilinx* (*Core Generator*). Estas memórias apresentam a vantagem de isolar os domínios de relógio da rede Ethernet e da ligação USB, podendo a memória ser escrita aquando da recepção de mensagens e lida para transmissão das mesmas via USB, utilizando sinais de relógio completamente independentes. Desta forma, é possível evitar eventuais problemas de sincronismo. Para além disso, a FIFO encapsula todos os mecanismos de gestão da memória e permite optimizar a utilização de recursos pois apesar do tamanho das tramas Ethernet ser variável estas são armazenadas em posições consecutivas, ocupando toda a memória disponível e sendo o número de mensagens capazes de ser armazenadas dependente do tamanho das mesmas.

Os vários passos para criação das memórias utilizando o *Fifo Generator* encontram-se anexados a este documento. Estas foram criadas definindo sinais de relógio independentes para leitura e escrita e sendo implementadas à custa de *Block RAMs*. Para além disto, na memória de dados foi definido um barramento de dados para escrita de 8 bits e na memória de controlo um barramento de 32 bits. O tamanho destes barramentos está relacionado com a informação que é escrita em cada uma. A informação para escrita na memória de dados é disponibilizada num barramento de 8 bits proveniente do TEMAC. As memórias FIFO permitem ainda ter portos de leitura e de escrita com tamanhos diferentes. Os diferentes campos a serem escritos na memória de controlo têm uma dimensão de 32 bits, no entanto,

para transmissão via USB são lidas palavras de 8 bits. Ambas as memórias apresentam capacidade para armazenar 16384 bytes e por este motivo foi adicionado um sinal de 14 bits designado *Read Data Count* que indica o número de bytes disponíveis para leitura em cada uma das memórias. Esta capacidade permite armazenar 256 mensagens de tamanho mínimo e aproximadamente 10 mensagens de tamanho máximo. Quanto à memória de controlo permite armazenar informação relativa a cerca de 1365 mensagens.

4.3.4 Escrita na Memória

Na figura 4.6 é possível observar o diagrama de blocos ilustrativo da forma como as mensagens recebidas são armazenadas. Para além das memórias e dos sinais provenientes do TEMAC e do módulo responsável pela realização do *time-stamping* é possível observar dois blocos que correspondem a máquinas de estados que controlam a escrita em cada uma das memórias.

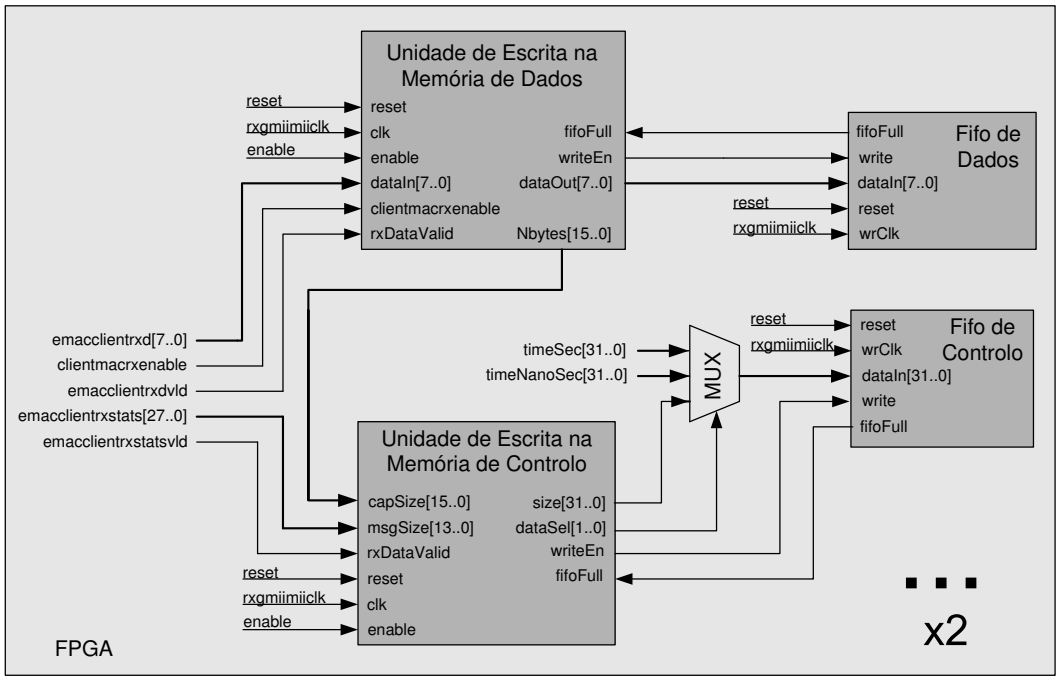


Figura 4.6: Diagrama de blocos da recepção e armazenamento das mensagens.

Escrita na Memória de Dados

A unidade de escrita na memória de dados, como o próprio nome indica, é uma máquina de estados responsável pela escrita do conteúdo da mensagem na memória destinada aos dados. Quando o primeiro byte de cada mensagem é recebido, se a captura se encontrar activa, à medida que os dados vão sendo disponibilizados pelo TEMAC, vão sendo armazenados na memória. Para além disso, uma vez que é possível limitar o número de bytes capturados (por opção do utilizador ou por falta de espaço na memória) esta unidade é também responsável pela contagem do número de bytes armazenados.

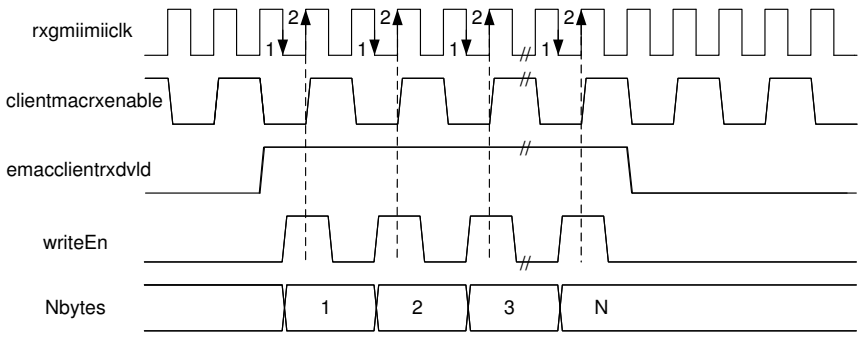


Figura 4.7: Diagrama temporal dos sinais mais relevantes para escrita na memória de dados.

Implementação

A máquina de estados actua nos flancos descendentes do sinal de relógio (*rxgmiimiiclk*). Se houver dados válidos no barramento (*emacclientrxdvld*='1') e o sinal *clientmacrxenable* se encontrar a '0' activa o sinal de escrita na memória (1), sendo a operação efectuada no flanco ascendente imediatamente a seguir (2). Quando o sinal *clientmacrxenable* se encontra a '1' o sinal de escrita na memória é desactivado. Em simultâneo com tudo isto, sempre que o sinal de escrita na memória é activado o número de bytes recebidos é incrementado e os dados presentes na saída do TEMAC são colocados no barramento de dados para escrita na memória. Tudo isto se encontra representado na figura 4.7 correspondendo os flancos ascendentes identificados com setas à escrita na memória e os flancos descendentes à activação do sinal de escrita na memória.

Escrita na Memória de Controlo

A unidade de escrita na memória de controlo é uma máquina de estados responsável pela escrita de toda a informação associada à mensagem recebida. A escrita desta informação na memória efectua-se em três fases, correspondendo cada uma delas à escrita de uma das entradas do *multiplexer* que se encontra na figura 4.6, isto é, o segundo em que a mensagem foi recebida, o nanosegundo e a informação relativa ao tamanho (da mensagem e da captura).

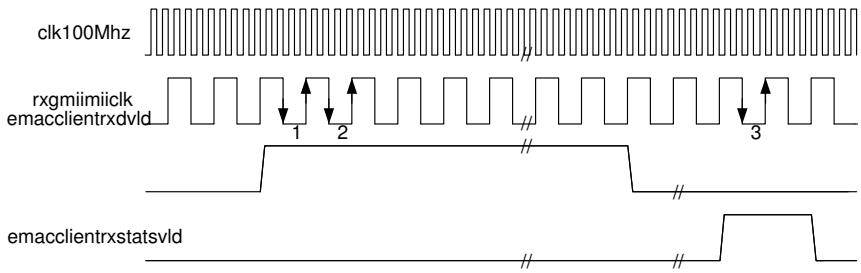


Figura 4.8: Diagrama temporal dos sinais mais relevantes para escrita na memória de controlo.

Implementação

Para escrita na memória de controlo e actualização da máquina de estados são usados exactamente os mesmos flancos do sinal de relógio que no caso anterior (figura 4.8). Desta

forma, ao serem detectados dados válidos no barramento (*emacclientrxdvld*='1'), o campo com o instante em segundos em que a mensagem foi recebida é colocado no barramento de dados para escrita na memória (1). No ciclo de relógio seguinte é escrito o campo que contém o nanosegundo de recepção da mensagem (2). Finalmente quando o vector de informação da mensagem se encontrar disponível (*emacclientrxstatsvld*='1') é escrito o tamanho da trama bem como o número de bytes capturados (3). De referir que este vector apenas é disponibilizado após a recepção da trama pelo que o número de bytes capturados (contabilizados pela unidade responsável pela escrita na memória de dados) também já se encontra disponível. Como o barramento de escrita de dados na memória é de 32 bits são utilizados os 16 bits menos significativos para armazenar o número de bytes capturados e os restantes 16 mais significativos para armazenar o tamanho da trama recebida.

4.3.5 Transferência de Informação da FPGA para o PC

Após as mensagens serem recebidas e toda a informação associada estar disponível torna-se necessário proceder ao seu encaminhamento para um computador permitindo uma posterior análise do tráfego capturado.

Interface utilizada

As principais possibilidades consideradas foram a utilização de uma interface Ethernet ou USB. A maior parte dos computadores existentes incluem apenas uma placa de rede, por este motivo, não seria possível utilizar o *sniffer* ligado a um PC com requisitos de rede para outros fins ou inclusivamente num dos nós da rede, sendo necessário um computador adicional. Por outro lado, a interface USB é amplamente utilizada e cada computador disponibiliza habitualmente várias portas, tornando a utilização da ferramenta muito mais flexível. Para além disso, e mais importante é a largura de banda disponibilizada por cada um dos protocolos. Como se pretende que o *sniffer* seja capaz de analisar ligações *fast* Ethernet em modo *full-duplex* convém que a largura de banda disponível para transferência de informação seja pelo menos o dobro da largura de banda oferecida pelas ligações *fast* Ethernet. Por este motivo, a utilização de uma ligação *fast* Ethernet é de todo desapropriada. A solução passaria pela utilização do padrão *gigabit* Ethernet, contudo, entre as várias placas de desenvolvimento disponíveis nenhuma dispõe de uma interface *gigabit* Ethernet e apenas os computadores mais recentes vêm equipados com placas de rede capazes de suportar este padrão. Por outro lado, segundo a norma USB, este protocolo oferece uma largura de banda útil para transmissão de dados até 425Mbps, motivo pelo qual foi a interface escolhida.

O microcontrolador EZ-USB FX2 USB da *Cypress* [Cyp07] é um circuito integrado que contém o *transceiver USB*, a interface série e um microcontrolador 8051. Para além disso, é capaz de operar no modo *high-speed* que permite taxas de transferência até 480Mbps e já se encontra na placa de desenvolvimento. Por estes motivos foi o componente utilizado para ligar à FPGA, permitindo transferir os dados desta via USB para um PC.

Na figura 4.9, retirada do *datasheet* do componente, encontra-se representada a configuração adoptada e que permite tirar maior proveito da largura de banda disponível. Esta configuração permite utilizar até 4 canais de comunicação (*EndPoints - EP*), podendo cada um ser utilizado para leitura ou escrita, estando associada a cada um deles uma memória FIFO. Neste caso, apenas foi utilizado um dos canais para envio da informação. Assim sendo, do ponto de vista da FPGA a transferência de dados via USB consiste simplesmente na es-

crita destes na memória FIFO do controlador, sendo da responsabilidade do PC a leitura dos mesmos.

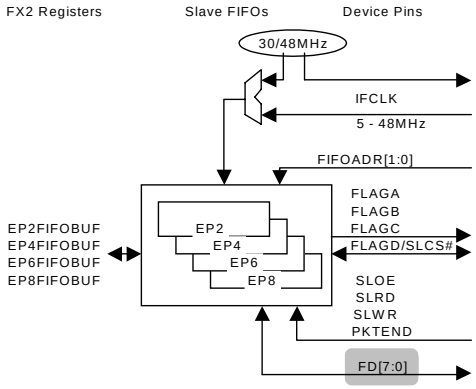


Figura 4.9: Diagrama de blocos de acesso ao controlador USB (retirado de [Cyp07]).

Implementação

Para evitar eventuais problemas de sincronismo devido a atrasos entre o controlador USB e FPGA, o sinal de relógio utilizado pela memória (IFCLK), em vez de ser um relógio interno de 30 ou 48Mhz, foi fornecido pela FPGA. Neste caso, uma vez que o barramento de dados (FD[7:0]) é de 8 bits optou-se por utilizar um relógio de 25MHz, obtendo-se desta forma uma taxa de transmissão de 200Mbps, ou seja, o dobro da largura de banda de uma ligação *fast* Ethernet. De seguida é efectuada uma apresentação sucinta dos restantes sinais de interface entre a FPGA e o controlador.

Uma vez que apenas é utilizado um canal de comunicação para transmissão de dados, o sinal (*FIFOADR[1:0]*), que permite seleccionar um dos 4 canais apresenta um valor fixo ("10"), correspondendo à utilização do EP6. O sinal *SLWR* é usado para activar a escrita na FIFO do controlador. Os sinais *SLRD* e *SLOE* estão relacionados com a leitura da FIFO, pelo que neste caso não são utilizados. Uma vez que a informação é enviada para o PC em blocos de 512 bytes, o sinal *PKTEND* serve para sinalizar um pacote como completo, permitindo desta forma enviar pacotes cujo tamanho é inferior a 512 bytes. Convém referir que a memória FIFO se encontra organizada em 4 blocos de 512bytes. Finalmente, das várias *flags* disponíveis, apenas é utilizada uma, servindo para detectar situações em que a memória do controlador se encontra totalmente ocupada, não sendo possível transferir informação para esta.

Leitura da Memória e Envio via USB

O envio dos dados via USB consiste então na transferência destes das memórias de dados e de controlo, nas quais foram armazenados durante a recepção, para a memória FIFO do controlador USB, sendo necessário multiplexar a leitura das várias memórias. Esta situação encontra-se representada no diagrama da figura 4.10.

Para facilitar o processamento dos dados do lado do PC estes são enviados da FPGA já no formato em que são armazenados em ficheiro. A única diferença em relação à forma em que se encontram na FPGA prende-se com o tamanho dos campos que contêm o tamanho

da mensagem e o número de bytes capturados. Por este motivo, é necessário extender o comprimento destes campos de dois para quatro bytes, sendo por isso adicionados a cada campo dois bytes com o valor zero. Esta é a justificação para a utilização de um multiplexer com 3 entradas (figura 4.10), tendo uma delas o valor "0x00".

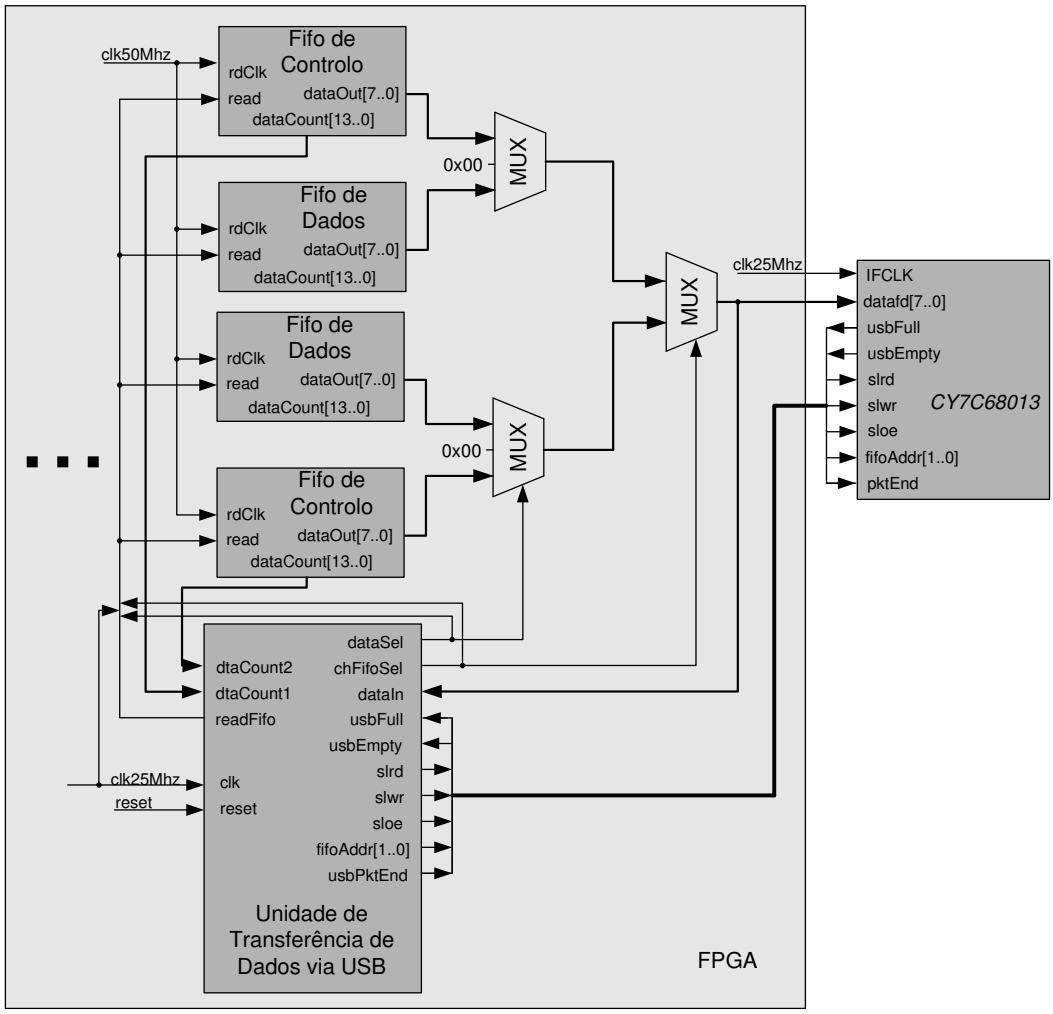


Figura 4.10: Diagrama de blocos da transmissão de dados via USB.

Para que cada mensagem possa ser transferida via USB ininterruptamente a sua transmissão apenas se inicia quando a sua recepção via Ethernet estiver concluída. Esta situação é detectada com base no número de bytes presentes na memória de controlo. Uma vez que o tamanho da mensagem é a ultima coisa a ser escrita na memória, a partir deste momento é certo que existe uma mensagem completa e como tal pronta a ser enviada. A partir deste momento a informação começa a ser lida da memória e escrita na FIFO USB. A transmissão da mensagem inicia-se transferindo a informação temporal armazenada na mensagem de controlo. Segue-se o envio do número de bytes capturados, devendo também esta informação ser armazenada localmente para determinar o número de bytes que devem ser lidos da memória de dados. O último campo a ser enviado desta memória é o tamanho total da mensagem.

Após toda a informação associada à mensagem ter sido copiada para a FIFO USB é

necessário proceder ao envio da própria mensagem. Para isso, é efectuado um número de leituras da memória de dados igual ao número de bytes capturados e respectiva escrita na FIFO USB.

Após a transmissão de cada uma das mensagens é analisado o estado das memórias de controlo. Caso alguma delas tenha mensagens prontas a ser transferidas, começa a ser enviada a mensagem da memória que apresenta mais mensagens prontas para transmissão. Caso não existam mensagens prontas para ser transmitidas e se a captura tiver sido terminada, o sinal *PKTEND* é activado forçando a que os dados presentes na FIFO USB sejam enviados para o PC mesmo que esta quantidade seja inferior a 512 bytes.

Antes de efectuar transferência de dados para a FIFO do controlador USB é necessário ter em atenção se esta se encontra cheia. Por este motivo, são necessários dois sinais de relógio, tendo sido utilizados um de 25MHz e outro de 50MHz. O sinal mais lento é utilizado nos flancos descendentes para actuar nos sinais de escrita e leitura das memórias (1). O flanco ascendente é usado para efectuar a escrita na FIFO do controlador USB (3). É no entanto necessário um outro flanco para leitura da informação armazenada nas memórias (2), sendo utilizado o flanco descendente do sinal de 50MHz quando o sinal de 25Mhz se encontra no nível lógico '0'. Esta situação encontra-se descrita na figura 4.11.

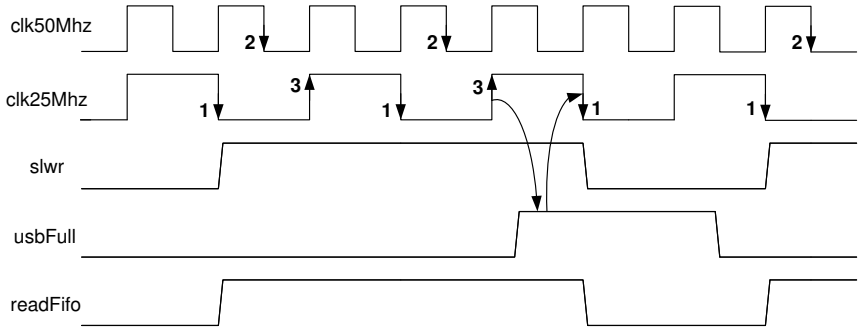


Figura 4.11: Diagrama temporal dos sinais mais relevantes para envio dos dados via USB.

Nesta situação são necessários dois sinais de relógio pois caso a escrita na FIFO do controlador USB fosse efectuada no mesmo flanco em que os sinais de controlo são actuados, o circuito não funcionaria correctamente. Uma vez que a memória só pode ficar cheia após uma escrita é necessário que a actualização dos sinais de controlo seja atrasada, garantido que o sinal *full* se encontra actualizado antes de activar o sinal para nova escrita (figura 4.11).

Apesar de no exemplo da figura 4.11 as formas de onda dos sinais de leitura da memória e escrita na FIFO do controlador USB serem iguais, estes não podem ser substituídos por um único sinal pois esta situação nem sempre se verifica. É disso exemplo a situação em que se escreve o valor 0x00 (para extender o comprimento dos campos com o tamanho) na FIFO do controlador USB, não sendo neste caso o valor lido da memória.

4.4 Modelação e Síntese

A modelação do *sniffer* foi efectuada utilizando a linguagem de descrição de *hardware* VHDL, tendo sido utilizado como ambiente de desenvolvimento o ISE (*Integrated Synthesis Environment*) [Xil08b] 9.2i da Xilinx. A forma como o projecto foi organizado encontra-se representada na figura 4.12.

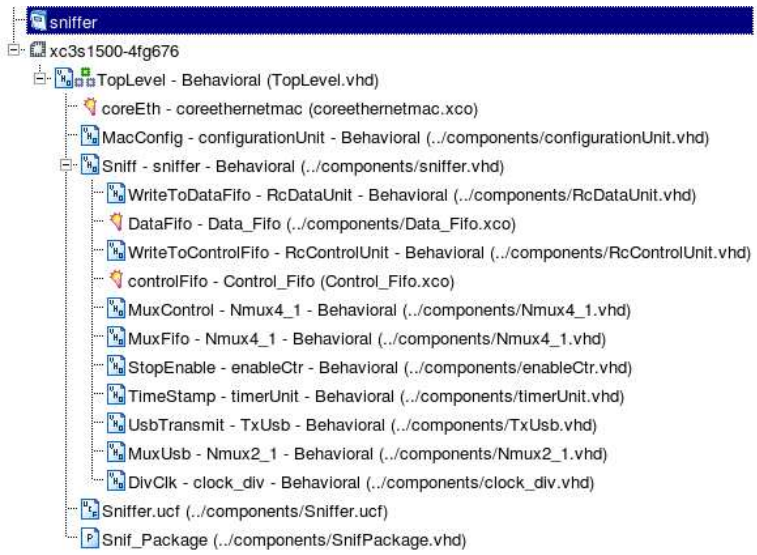


Figura 4.12: Organização hierárquica do projecto no ISE.

No topo da hierarquia existe um ficheiro designado *TopLevel.vhd*, no qual é feita a instanciação do *sniffer* propriamente dito, do TEMAC e da máquina de estados para configuração do mesmo. No ficheiro *sniffer.vhd* encontram-se instanciados todos os componentes que implementam a lógica de processamento das mensagens e envio via USB. Com base no nome de cada um dos ficheiros torna-se intuitiva a sua associação a cada um dos blocos descritos nas secções anteriores e representados nas figuras 4.2, 4.6 e 4.10. Apenas os blocos *StopEnable* e *DivClk* não foram referenciados anteriormente, estando relacionados com detalhes de implementação. O primeiro é responsável por activar/desactivar o funcionamento do *sniffer*, com base nos botões de interface com o utilizador. O segundo serve para a partir de um sinal de relógio de 100Mhz obter os sinais de 50Mhz e de 25Mhz utilizados pelos vários blocos constituintes do circuito. Existe ainda um ficheiro designado *sniffer.ucf* no qual são definidos todos os pinos da FPGA que interagem com o exterior, isto é, para ligação ao Phy, aos botões de pressão e ao controlador USB. Foi também criada uma *package* na qual se encontram declaradas as interfaces de alguns componentes como o TEMAC e as FIFOs. É também nesta *package* que o utilizador pode definir o número máximo de bytes a serem capturados de cada uma das mensagens recebidas.

O *sniffer* foi sintetizado e implementado numa FPGA de baixo custo, uma XC3S1500 da família Spartan3 da Xilinx. O relatório final da síntese pode ser visto na imagem 4.13.

O atraso máximo introduzido pelo seu caminho crítico permite uma frequência máxima de funcionamento de 57Mhz, ocupa aproximadamente 2789 células lógicas, o que corresponde a 20% das células disponíveis, sendo de salientar que todas as block RAMs disponíveis se encontram utilizadas. A sua implementação em ASIC corresponderia à utilização de cerca de 2,120,985 portas lógicas.

```
Final Synthesis Report
=================================================
Selected Device :xc3s1500-4fg676
Device utilization summary:
N° of Slices: 2789 out of 13312 (20%)
N° of Slice Flip-Flops: 3587 out of 26624 (13%)
N° of 4 input LUTs: 4174 out of 26624 (15%)
N° of Bonded IOBs: 54 out of 487 (11%)
N° of Global CLKs: 7 out of 8 (87%)
N° of Block RAMs: 32 out of 32 (100%)
Timing Summary:
Speed Grade: -4
Min. period: 17.521ns (Max. Frequency: 57.075MHz)
Min. input arrival time before clock: 9.337ns
Max. output required time after clock: 9.922ns
Maximum combinational path delay: 12.338ns
```

Figura 4.13: Relatório da síntese gerado pelo ISE.

4.5 Ligação da Ferramenta à Rede de Comunicação

Considerando a finalidade com que o *sniffer* foi concebido, é de todo desapropriado que a sua introdução na rede de tempo-real provoque alterações no comportamento lógico e temporal da mesma, devendo a sua inclusão, ser o mais transparente possível. A solução habitualmente utilizada quando se lida com *sniffers* Ethernet consiste na introdução de um *hub* no ramo da rede em que se pretende analisar o tráfego. Desta forma, todo o tráfego que o atravessa é também reencaminhado para o analisador (figura 4.14).

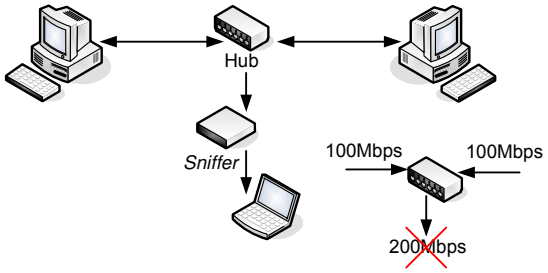


Figura 4.14: Inserção do *sniffer* na rede com recurso a um *hub*.

Esta solução é no entanto desapropriada para o fim em questão pois para além da perturbação causada na rede, devido às colisões geradas pela introdução do *hub*, a chegada de tráfego a uma taxa de $100Mbps$ simultaneamente nas duas portas do *hub*, não permite que o registo do instante em que cada mensagem é recebida seja efectuado correctamente uma vez que não é possível reencaminhar os dados para o *sniffer* a uma taxa de $200Mbps$. Isto obriga, no caso de uma ligação *full-duplex*, à existência no *sniffer* de dois portos que permitam registar simultaneamente os instantes de recepção das mensagens que circulam nos 2 sentidos.

Uma solução alternativa consiste em duplicar o sinal para 2 barramentos distintos (o próprio barramento da rede e outro para ligação ao *sniffer*) como é possível observar na figura 4.15. Isto pode ser feito com recurso a um *TAP* Ethernet também conhecido por *Splitter*. Existem no mercado diversos *TAPs* com características distintas [Dat08a] [Dat08c] [Leo07] que permitem, funcionando a $100Mbps$, obter latências inferiores ao tempo de um bit. Estes produtos são comercializados por várias empresas, tais como Net Optics, Inc. [Opt08],

Network Critical [Lim08], Finisar [Fin08], DataCom Systems Inc. [Dat08b], sendo, em média, o seu valor próximo de 300€.

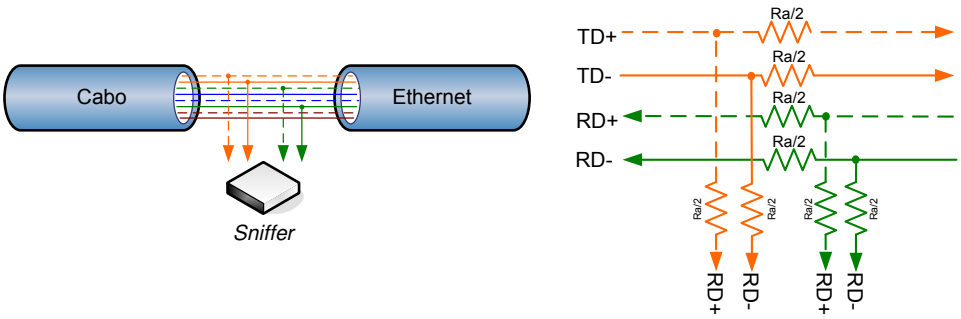


Figura 4.15: Inserção do *sniffer* na rede com recurso a um *TAP*.

Tendo em conta o preço do equipamento e que o mesmo será utilizado para testes em pequenas redes, em ambiente académico, optou-se pela construção de um *TAP* passivo. Obviamente, cada um dos sinais duplicados apresenta no mínimo uma atenuação de 3db, não sendo no entanto crítico para o correcto funcionamento das redes em questão. Os cabos Ethernet utilizados são constituídos por 4 pares entrançados, sendo que apenas dois deles são utilizados, um para recepção e outro para transmissão. Como é possível verificar na figura 4.15, a construção do *TAP* consiste na ligação de 2 fios a cada um dos pares entrançados utilizados. Foram introduzidas resistências para adaptação (Ra), sendo o seu valor obtido com base num compromisso entre o coeficiente de reflexão e a atenuação no sinal.

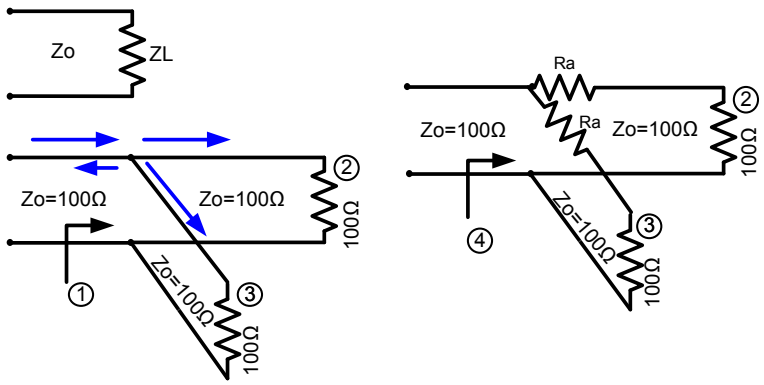


Figura 4.16: Adaptação de impedâncias no meio de transmissão.

O coeficiente de reflexão numa linha com impedância característica *Z0* e terminado com uma carga *ZL* é dado por $\rho = (ZL - Z0)/(ZL + Z0)$. Na situação em que o coeficiente de reflexão não é nulo, é gerada uma onda igual à soma da onda incidente com a onda reflectida, podendo originar erros. Para um correcto funcionamento é então necessário que impedância da carga seja igual à impedância característica do cabo. Analisando a figura 4.16 e considerando que nos pontos 2 e 3 a linha está conectada a um equipamento e portanto completamente adaptada, a onda que lá chega não sofre qualquer reflexão. O ponto problemático é onde a onda incidente encontra as duas linhas em paralelo. Neste caso, a impedância vista a partir

do ponto 1 será o paralelo das impedâncias características das duas linhas, ou seja, 50Ω, originando um coeficiente de reflexão de $-1/3$. Este coeficiente de reflexão pode ser anulado com a introdução das resistências Ra. Se $Ra = 100\Omega$, a impedância vista a partir do ponto 4 será $100\Omega((100 + 100)//(100 + 100))$, estando a linha completamente adaptada. Contudo, uma vez que a potência dissipada na resistência é proporcional ao seu valor e nesta situação apenas 25% da potência incidente é entregue à carga, optou-se, sacrificando ligeiramente o coeficiente de reflexão, por baixar o valor das resistências para 66Ω, diminuindo a potência dissipada nas mesmas. O resultado final pode ser visto na figura 4.15 sendo que em vez de se utilizar uma única resistência, foram usadas duas de 33Ω, uma em cada linha do par entrançado.

Capítulo 5

Componentes de Software do Sniffer Desenvolvido

5.1 Introdução

Neste capítulo é efectuada uma descrição de todo o *software* constituinte do *sniffer*. Este encontra-se dividido em duas secções. A primeira está relacionada com a transferência de dados via USB, sendo descrito o *software* implementado do lado do PC para recepção e armazenamento da informação e também o *firmware* utilizado no controlador USB que se encontra ligado à FPGA. Na secção seguinte é descrita a forma como os dados adquiridos podem ser tratados e analisados.

5.2 Transferência via USB

O módulo utilizado para interface com a FPGA foi um microcontrolador EZ-USB FX2 USB da *Cypress* (*CY7C68013-100AC*). É um circuito integrado que contém o *transceiver USB*, a interface série e um microcontrolador 8051. O seu diagrama de blocos encontra-se representado na figura 5.1. Para além disso, suporta ligações USB 2.0 capazes de operar em modo *high-speed* e encontra-se já integrado na placa de desenvolvimento utilizada.

Segundo a norma USB, as ligações USB 2.0 em modo *high-speed* permitem obter taxas de transferência até 480Mbps. No entanto, considerando todo o *overhead* de informação de

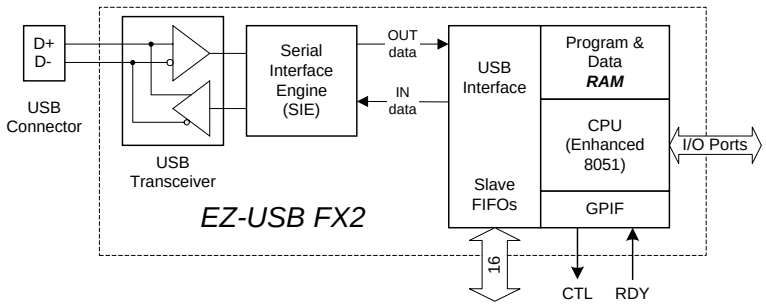


Figura 5.1: Diagrama de blocos do módulo USB utilizado (retirado de [Cyp07]).

controlo própria do protocolo este oferece uma largura de banda útil no máximo de 425Mbps. Este valor é no entanto fortemente dependente do *host* do lado do PC, sendo este responsável por coordenar toda a transferência de informação. Na prática verifica-se que o desempenho conseguido depende fortemente da capacidade de resposta do PC efectuar a leitura da informação presente no controlador USB.

O protocolo USB define quatro tipos de transferência de dados para dispositivos. Devido à grande quantidade de informação a transferir foi utilizado o modo *bulk*. Este modo é adequado para a transferência de grandes quantidades de dados, permitindo a transmissão com garantias que não existirão erros. Por este motivo poderá ser necessário repetir a transmissão de dados que tenham sofrido erros, não sendo portanto dadas garantias em termos de largura de banda. O modo *bulk* é o menos prioritário, utilizando a parte da trama deixada disponível pelos equipamentos que funcionam nos restantes modos. No entanto, se o barramento não se encontrar partilhado por outros dispositivos este modo de funcionamento pode ser o que permite obter maiores taxas de transferência. Por este motivo é importante que não se encontrem outros dispositivos, como por exemplo uma *webcam*, ligados via USB ao PC responsável pela recepção dos dados capturados.

5.2.1 *Firmware* para Transmissão de Dados da FPGA

Como é possível verificar na figura 5.1, a entrada e saída de dados no processador FX2 efectua-se com recurso a memórias do tipo FIFO. De forma a manter as elevadas taxas de transferência é habitualmente utilizada alguma lógica externa para escrita e leitura directamente nas memórias sem qualquer intervenção do microcontrolador do módulo FX2. Desta forma, como já foi visto no capítulo anterior, do ponto de vista da FPGA, este comporta-se como sendo uma simples memória FIFO.

Uma vez que o microcontrolador 8051 não desempenha qualquer função de processamento, o *firmware* necessário para a utilização do módulo corresponde apenas à configuração de alguns registos com o objectivo de:

- Definir um clock de 48Mhz para o processador (CPUCS=0x12);
- Colocar o módulo no modo de operação *slave FIFO*, ou seja, sem processamento do microcontrolador, utilizando um clock externo para sincronizar a escrita na FIFO (IFCONFIG=0x03);
- Activar o *End Point 6* para transmissão de dados no modo *bulk* e no modo *quad buf.* Desta forma, a memória é organizada em quatro blocos de 512 bytes (EP6FIFOCFG = 0x0C);
- Configurar todos os sinais de controlo no modo *active high*, ou seja, activos no nível lógico '1' (FIFOPINPOLAR=0x3F);
- Definir o tamanho com que os pacotes são enviados para o PC. Neste caso foram definidos pacotes de 512 bytes, tendo-se verificado que com este valor se alcança as taxas de transferência mais elevadas (EP6AUTOINLENH = 0x02 e EP6AUTOINLENL = 0x00).

Convém também salientar que entre a escrita de alguns destes registos é necessário introduzir um atraso para sincronização. Os registos entre os quais este atraso é necessário encontram-se especificados no *datasheet* do componente [Cyp07].

O código para configuração do micro-controlador foi construído em linguagem C, sendo baseado em alguns exemplos retirados de [Wie08]. Aqui é também possível encontrar todas as ferramentas utilizadas para compilar e carregar o *firmware* no microcontrolador. Para isso foi criado num ficheiro designado *firmware.c* bem como uma *makefile*, devendo encontrar-se ambos no mesmo directório. Assim, basta executar o comando *make* para gerar o ficheiro de configuração e o comando *make send* para o carregar no microcontrolador.

Para compilação do código foi utilizado o compilador *sdcc*, sendo necessário incluir os ficheiros fx2regs.h e fx2regs.inc. No exemplo que se segue estes encontram-se num directório designado *include*.

```
>> sdcc -mmcs51 -I./include firmware.c
```

Após a compilação do programa é gerado um ficheiro com a extensão *.ihx* sendo utilizada a aplicação *CycFX2Prog*, também disponível em [Wie08] para carregar o ficheiro no microcontrolador:

```
>> cycfx2prog prg:firmware.ihx run
```

5.2.2 Recepção de Dados no PC

Para recepção de dados no PC foi criada uma aplicação que deve ser colocada em execução antes de se iniciar a captura no *sniffer*. Esta recebe como parâmetro de entrada o nome para o ficheiro binário em que será armazenada a captura. Este aplicação é responsável pela recepção dos dados provenientes através da ligação USB e escrita dos mesmos para um ficheiro binário compatível com o *Wireshark*.

Antes do PC começar a efectuar os pedidos de envio de dados é necessário verificar se o dispositivo se encontra conectado a este. Para isso é necessário varrer os barramentos USB, percorrendo todos os dispositivos até que seja encontrado o dispositivo em questão. Este é reconhecido com base no identificador do fabricante (0x4b4) e no identificador do produto (0x8613). Para isso são utilizados dois ponteiros, enquanto um é utilizado para percorrer uma lista ligada dos vários barramentos, o outro serve para percorrer a lista ligada dos dispositivos presentes no barramento. Informações detalhadas podem ser encontradas em [Ceb08].

Após detectado o dispositivo procede-se então ao pedido de envio dos dados da FPGA para o PC, sendo estes armazenados num ficheiro do tipo *libpcap*. A estrutura deste tipo de ficheiros encontra-se representada na figura 5.2 e é o tipo de ficheiro criado pelas bibliotecas de captura (*libpcap e winpcap*) para comunicação com o *Wireshark*.

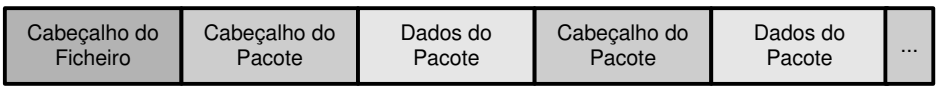


Figura 5.2: Formato do ficheiro Libpcap.

Nesta é possível verificar que antes dos dados propriamente ditos existe um cabeçalho no qual são especificadas as características do ficheiro e que irão permitir que este seja interpretado pelo *Wireshark*. Desta forma, antes de começar a receber os dados e a escrevê-los para o ficheiro é necessário a criação deste cabeçalho. De seguida é apresentado cada um dos seus campos e o valor atribuído a cada um deles:

- Constante para definição do formato do ficheiro (*Little-Endian*): 0xa1b23c4d;

- Versão do formato do ficheiro (parte inteira): 2;
- Versão do formato do ficheiro (parte fraccionaria): 4 (Este campo em conjunto com o anterior definem a actual versão que é a 2.4);
- Correcção em segundos entre o horário GMT e o horário local: 0;
- Resolução temporal: 9 (Nanosegundos);
- Tamanho máximo dos pacotes capturados: 1518 (Tamanho máximo das tramas Ethernet);
- Tipo de ligação de dados (Ethernet, Token Ring, FDDI ...): 1 (Ethernet).

Após este cabeçalho segue-se toda a informação relativa às tramas capturadas bem como o conteúdo das mesmas. A informação ao ser enviada da FPGA vêm já no formato compatível com o ficheiro criado. Desta forma, o processamento no PC é simplificado e a largura de banda disponível é utilizada na totalidade para transferência de informação, não sendo necessário definir um protocolo de comunicação específico e que aumentaria o *overhead* de informação próprio do protocolo. A informação de controlo proveniente da FPGA e que corresponde ao cabeçalho de cada trama recebida é constituída por:

- Tempo em segundos (32 bits);
- Desfazamento em nanosegundos dentro do segundo em questão (32 bits);
- Número de octetos do pacote armazenados no ficheiro (32 bits);
- Tamanho original do pacote capturado (32 bits);

Após esta informação segue-se o conteúdo das mensagens capturadas.

Implementação

Para recepção dos dados é necessário que a biblioteca *libusb* esteja instalada no sistema. O código para recepção dos dados foi construído em linguagem C, tendo para isso sido criado um ficheiro designado *usbread.c*. Este pode ser compilado, utilizando o compilador *gcc* da seguinte forma:

```
>> gcc -lusb usbread.c -o usbread
```

A recepção da informação efectua-se com recurso a uma função que executa um pedido de leitura de um determinado número de bytes e à qual é passado um determinado *time-out*. Em caso de sucesso esta retorna o número de bytes recebidos, ou um valor negativo caso ocorra algum erro.

Foram também efectuados vários testes, variando a quantidade de bytes requeridos em cada leitura, tendo-se verificado que as maiores taxas de transferência são conseguidas quando este valor é 65536, ou seja, 64Kbytes.

Limitações

Os primeiros testes efectuados permitiram obter taxas de transferência de informação próximas dos 240Mbps. Esta taxa foi no entanto conseguida sem qualquer processamento dos dados recebidos, ficando estes armazenados na memória RAM do PC. No entanto, as capturas efectuadas podem atingir dimensões consideráveis, sendo necessário armazenar a informação no disco. Efectuando a escrita dos dados recebidos via USB para um ficheiro verifica-se um decréscimo na largura de banda média utilizada para aproximadamente 130Mbps, devido aos tempos de bloqueio de escrita no ficheiro. Esta tarefa em conjunto com todas as outras que se encontram habitualmente em execução num PC e a possibilidade de ocorrerem retransmissões devido à ocorrência de erros contribuem para que a taxa de transferência conseguida se encontre muito abaixo do valor teórico esperado.

5.3 Tratamento de Dados

Uma vez que os ficheiros são armazenados num formato reconhecido pelo *Wireshark*, a análise dos dados pode ser efectuada com recurso a esta aplicação. O *Wireshark* é uma ferramenta que pode ser utilizada livremente e que se encontra disponível para vários sistemas operativos. Para além disso, permite descodificar mais de 750 protocolos de comunicação, podendo ser adicionados novos protocolos por qualquer programador. O aspecto de uma captura realizada com o *sniffer* e analisada com o *Wireshark* encontra-se na figura 5.3.

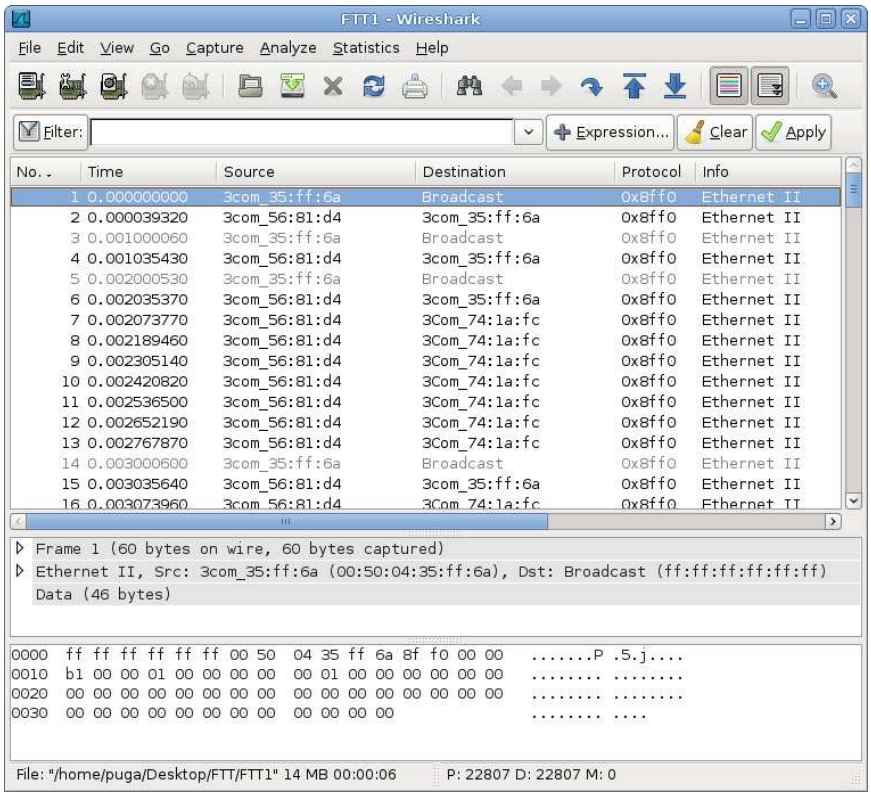


Figura 5.3: Aspecto de uma captura analisada com o *Wireshark*.

Esta ferramenta apresenta algumas funcionalidades que podem ser exploradas. Permite a

aplicação de filtros sobre as capturas efectuadas, permitindo mostrar apenas alguns tipos de mensagens, ou apenas as mensagens com determinado endereço de origem ou destino. Por este motivo, a aplicação de filtros na FPGA, limitando a captura poderá ter apenas interesse de forma a reduzir a largura de banda utilizada na ligação USB e o tamanho dos ficheiros criados. No entanto, uma vez capturadas todas as mensagens, o utilizador tem a possibilidade de seleccionar as que pretende analisar.

Para além disso, é possível alterar também a forma como o *time-stamp* das mensagens é exibido. Este podem ter como referência o instante de recepção da primeira mensagem, ser referenciado em relação à mensagem capturada imediatamente antes, ou, caso sejam aplicados filtros, em relação à mensagem exibida anterior. Convém também ter em atenção que a ordem pela qual as mensagens aparecem no ficheiro e são exibidas no *Wireshark* poderá não ser a ordem pela qual foram capturadas. No entanto, o *Wireshark* permite listar as mensagens pela ordem em que foram capturadas.

Apesar de ser uma ferramenta bastante útil para o utilizador efectuar uma inspecção visual do que realmente se passa na rede, não reúne características que permitam caracterizar o tráfego capturado. Por este motivo, é importante que os dados possam ser exportados e utilizados em ferramentas de cálculo matemático e que permitam efectuar representações gráficas, tornando-se relativamente simples efectuar uma análise em termos probabilísticos e de pior caso do tráfego que circula na rede.

O *Wireshark* oferece a possibilidade de exportar os dados para ficheiros do tipo CSV (*Comma Separated Values*) e ficheiros de texto. Apesar do formato CSV ser suportado por muitas ferramentas de cálculo, entre as quais o *octave/matlab* e folhas de cálculo das ferramentas *Office*, a informação neles disponibilizados poderá não ser suficiente. No caso do protocolo FTT, uma vez que o *Wireshark* não faz a descodificação do mesmo, este tipo de ficheiro não permite identificar por exemplo o tipo das mensagens (figura 5.5). Para além disso, a forma como o campo com a informação temporal é apresentado não permite que sejam efectuadas operações aritméticas sobre este.

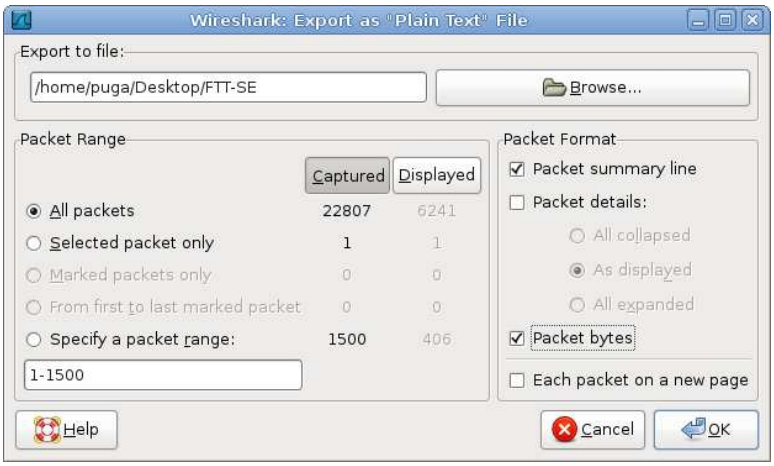


Figura 5.4: Exportação dos dados capturados para ficheiro de texto.

Por este motivo, e uma vez que poderá ser útil ter acesso não só a informação temporal mas também a informação que permita identificar cada uma das mensagens, foi criada uma aplicação baseada no *octave* que a partir de um ficheiro de texto permite gerar ficheiros com

mais informação que a disponível no formato CSV. A utilização de ficheiros de texto torna-se também desadequada pois estes geram um relatório demasiado detalhado sobre a captura, não podendo ser utilizados pelas folhas de cálculo. Quanto à sua utilização no *octave*, seria necessário todo o seu processamento sempre que se pretendesse importar dados. Os ficheiros de texto, deverão ser exportados de acordo com as opções que se encontram representadas na figura 5.4. Convém ainda referir que a forma como são exibidos os *time-stamps* das mensagens permanece quando os dados são exportados para ficheiro. Na figura 5.4 é também possível verificar que a exportação de dados pode contemplar todas as mensagens capturadas ou apenas as mensagens visualizadas e que resultam da aplicação de filtros.

O único campo estritamente necessário para criação de um ficheiro é o campo designado *Packet summary line* pois é neste que se encontra toda a informação temporal. O campo com os detalhes da trama *Packet Details* é de todo dispensável e caso seja incluído, o tamanho do ficheiro será maior, aumentando também o tempo de processamento do mesmo. Quanto ao campo de dados este pode ou não ser incluído, encontrando-se na figura 5.5 representado um excerto do ficheiro gerado em cada uma das situações. Mesmo caso o campo de dados não seja incluído este apresenta uma vantagem em relação ao formato CSV: a forma como o campo com informação temporal se encontra representado permite que seja directamente utilizado para a realização de cálculos.

```
==== Ficheiro CSV gerado a partir do wireshark:
"No.","Time","Source","Destination","Protocol","Info"
"1","0.000000000","3com_35:ff:6a","Broadcast","0x8ff0","Ethernet II"
"2","0.000039320","3com_56:81:d4","3com_35:ff:6a","0x8ff0","Ethernet II"
"3","0.001000060","3com_35:ff:6a","Broadcast","0x8ff0","Ethernet II"

==== Ficheiro gerado a partir de um ficheiro de texto sem campo de dados:
"No.","Time","Source","Destination","Protocol"
1 0.000000000 3com_35:ff:6a Broadcast 0x8ff0
2 0.000039320 3com_56:81:d4 3com_35:ff:6a 0x8ff0
3 0.001000060 3com_35:ff:6a Broadcast 0x8ff0

==== Ficheiro gerado a partir de um ficheiro de texto com campo de dados:
"No.","Time","Source","Destination","Protocol","FttType","Data"
1 0.000000000 00500435ff6a ffffffffffff 8ff0 0000 b1000001000000000001000000000000
2 0.000039320 0050045681d4 00500435ff6a 8ff0 0003 000200003d3d3d3d3d3d3d3d3d3d3d
3 0.001000060 00500435ff6a ffffffffffff 8ff0 0000 b2000001000000000001000000000000
```

Figura 5.5: Formato dos diferentes ficheiros criados a partir do *Wireshark*.

Caso as mensagens capturadas apresentem um grande volume de dados, os ficheiros resultantes da exportação poderão também ter tamanhos bastante consideráveis pelo que o tempo de processamento destes se pode tornar bastante longo. Uma solução para este problema seria a utilização de uma aplicação existente designada *editcap* que permite editar ficheiros do tipo *libcap*. A utilização do seguinte comando deveria limitar o tamanho do campo de dados de todas as tramas a 100 bytes.

```
>> editcap -s 100 -F nseclibpcap  ficheiroOriginal ficheiroEditado
```

No entanto, verificou-se que esta ferramenta nem sempre funcionou de forma correcta, tendo em algumas situações desaparecido cerca de 50% dos pacotes existentes no ficheiro

original. Uma solução alternativa passa por limitar a quantidade máxima de dados a capturar na própria FPGA.

Uma vez que se pretende dar principal atenção ao protocolo FTT, os ficheiros gerados, nas situações em que o campo de dados é exportado, para além de incluírem uma coluna com o protocolo (correspondendo o valor "8ff0" ao protocolo FTT). Apresentam também uma coluna na qual é especificado o tipo de trama FTT (*Trigger Message*: "0000"; *Asynchronous Status Message*: "0003") e uma coluna com os primeiros 16 bytes de cada trama FTT.

Para processamento dos ficheiros de texto exportados a partir do *Wireshark* foi criado um *script* designado *txt2ssv.m*. Este recebe como parâmetros de entrada o nome do ficheiro a ser processado e o nome do ficheiro no qual a informação deverá ser armazenada. Para isso basta no *octave* executar o seguinte comando:

```
>> txt2ssv('fileInName','fileOutName')
```

A importação dos dados presentes nos ficheiros criados para folhas de cálculo das ferramentas *Office* é efectuada da mesma forma que os ficheiros do tipo CSV. Quanto ao *octave*, a importação pode ser efectuada utilizando o seguinte comando:

```
>> [n,Time,Src,Dst,Prot,FttType,Data]=textread('fileInName',"%d %d %s %s %s %s %s")
```

A utilização dos dados no *octave* requer algum cuidado, nomeadamente no que diz respeito à forma como os tempos são apresentados. Mesmo que os pacotes sejam ordenados utilizando o *Wireshark* esta ordenação não permanece quando os dados são enviados para o ficheiro de texto, cabendo ao utilizador ordenar os dados se assim o entender.

O processamento dos dados no *octave* depende da forma como a sua exportação for efectuada. Para efectuar uma análise ao *jitter* de uma mensagem periódica é possível no *Wireshark* aplicar um filtro para que apenas esta mensagem seja exibida, e formatar o campo com o *time-stamp* para que este valor seja a diferença temporal em relação à mensagem anterior. Desta forma, os intervalos de tempo entre mensagens consecutivas já se encontram calculados. No *octave* basta então importar os dados, traçar o seu gráfico e eventualmente calcular a diferença entre o valor máximo e mínimo. Outra solução passa por exportar toda a informação para o ficheiro, aumentando a complexidade do processamento no *octave*. Nesta situação seria necessário importar os dados, ordenar todas as mensagens, calcular os intervalos de tempo entre as mensagens pretendidas e só depois seria possível traçar o gráfico e calcular o valor do *jitter*.

Desta forma, o *Wireshark* torna-se bastante flexível, permitindo exportar a informação no formato mais conveniente. Foi também criado um ficheiro designado *captureAnalyzer.m*, sendo um exemplo de uma aplicação baseada no *octave*. Este permitiu testar estas ferramentas, avaliar o desempenho do *sniffer* e caracterizar as mensagens periódicas próprias do protocolo FTT-SE. Alguns dos resultados obtidos são apresentados no capítulo seguinte.

Capítulo 6

Testes e Análise do Desempenho

6.1 Introdução

De forma a analisar o desempenho da ferramenta construída, esta foi submetida a vários testes. Estes tiveram como objectivo efectuar uma comparação entre os resultados obtidos fazendo a captura com recurso ao *sniffer* construído e com recurso a ferramentas baseadas em *software*, em particular o *Wireshark*. Neste contexto, foi analisada a interferência causada pela introdução da ferramenta na rede de comunicação e a capacidade de capturar mensagens. Para além disso, foi feita uma análise da precisão obtida nas medições temporais efectuadas. Em relação à ferramenta construída, pretende-se também analisar a capacidade de resposta da ligação USB face à taxa de utilização da ligação Ethernet. Finalmente, a ferramenta foi utilizada para analisar o desempenho do protocolo FTT-SE numa situação concreta envolvendo controlo distribuído. Neste capítulo são apresentados e analisados os resultados obtidos na realização de todos os testes mencionados.

6.2 Hardware *versus* Software

De forma a comparar os resultados obtidos efectuando a captura usando aplicações baseadas em *software* e utilizando *hardware* dedicado efectuaram-se duas montagens semelhantes. Primeiro, efectuou-se uma ligação entre dois computadores, sendo um responsável pela geração de pacotes (utilizando a aplicação packETH [Pac08]) e outro responsável pela captura dos mesmos utilizando o *Wireshark*. Em cada experiência foram gerados 500000 pacotes, encontrando-se o número de pacotes capturados indicado na quarta coluna da tabela 6.1.

De seguida, efectuaram-se exactamente os mesmos ensaios tendo sido introduzido o *sniffer* (recorrendo ao TAP) entre os dois computadores. Nesta situação foram efectuadas duas capturas em simultâneo, utilizado o *Wireshark* e o *sniffer* construído. Os resultados alcançados efectuando as capturas com ferramentas baseadas em *software* e *hardware* encontram-se respectivamente nas colunas cinco e seis da tabela 6.1.

Da análise dos resultados obtidos, a primeira conclusão que se pode retirar é que efectuando a captura com base em *software* são perdidos pacotes, sem que o utilizador seja alertado dessa situação. Por outro lado, utilizando *hardware* dedicado o número de pacotes capturados corresponde exactamente ao número de pacotes gerados.

Analisando os resultados com mais detalhe verifica-se ainda que a quantidade de pacotes perdidos diminui com o aumento do intervalo de tempo entre os mesmos. Desta forma, o

Condições (500000 pacotes)			Percentagem de pacotes capturados		
Período	Tamanho	Ocupação	Sem Tap	Com Tap	
(μs)	($Bytes$)	($Mbps$)	Software (%)	Software (%)	Hardware (%)
10	80	94	73.8	72.6	100
50	80	19	99.0	99.4	100
50	400	70	94.9	94.9	100
100	80	9	99.5	99.8	100
100	900	75	96.7	97.2	100
300	80	3	100	100	100

Tabela 6.1: Mensagens perdidas e interferência do TAP na rede de comunicação.

PC tem mais tempo para transferir e processar a informação, esvaziando as filas da placa de rede. A mesma situação verifica-se quando o tamanho das mensagens diminui uma vez que a quantidade de dados a ser transferidos é menor.

Verifica-se também que com a introdução na rede do *sniffer* desenvolvido o número de mensagens perdidas pela aplicação baseada em *software* permanece praticamente constante, na realidade verifica-se que por vezes este valor diminui. Por este motivo, é possível concluir que para as ligações utilizadas no laboratório a atenuação causada pelo TAP não tem qualquer interferência relevante no comportamento lógico da rede de comunicação.

6.2.1 Precisão Temporal

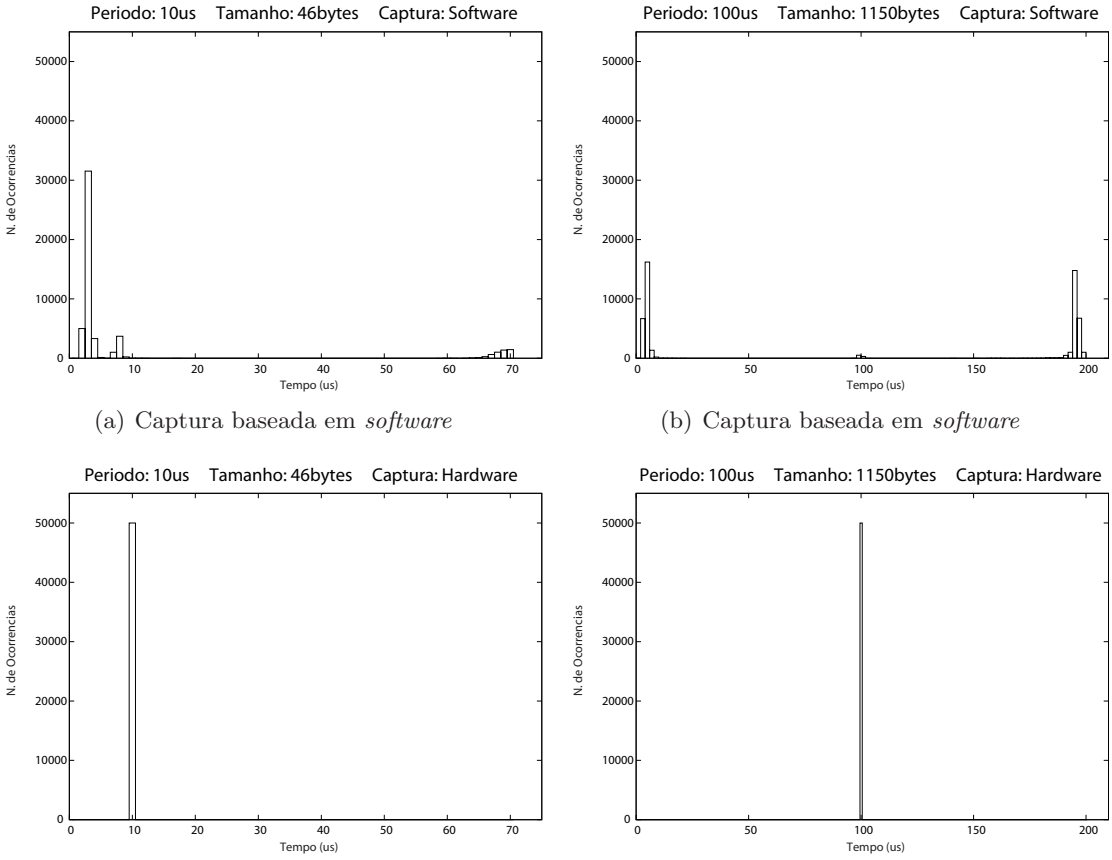
De forma a avaliar a precisão das medições efectuadas procedeu-se à injecção de tráfego nas ferramentas de captura. As várias experiências foram efectuadas com taxas de ocupação do meio de transmissão distintas. Para isso, não só foi alterado o intervalo de tempo entre mensagens (período) mas também o tamanho das mesmas. Para gerar tráfego foi também utilizado *hardware* dedicado e baseado no TEMAC, sendo usada numa outra FPGA responsável apenas pela geração de pacotes. Desta forma, os pacotes são gerados eliminando toda a incerteza temporal inerente ao funcionamento dos PCs. De acordo com as especificações presentes no manual de utilização do TEMAC, no pior caso o erro poderá atingir 120ns, resultante da sincronização entre os vários domínios de relógio do controlador e interacção com o Phy.

O resultado das capturas foi posteriormente exportado para o *octave*, no qual foi efectuada uma análise do tráfego capturado. Para isso, foram calculados os valores mínimo, médio e máximo do período bem como o desvio padrão e o *jitter* absoluto, ou seja, a diferença entre os valores máximo e mínimo. Todos estes valores encontram-se representados nas tabelas 6.2 e 6.3, correspondendo estas às capturas baseadas em *software* e *hardware*.

Para facilitar a análise dos resultados obtidos estes foram convertidos para a forma de histogramas. No entanto, nas figuras 6.1 e 6.2 apenas estão representadas quatro situações distintas. Isto deve-se ao facto dos resultados da captura baseada em *hardware* serem todos muito semelhantes e coerentes. No caso da captura com recurso ao *Wireshark* observaram-se duas situações distintas, sendo no entanto os resultados bastante semelhantes para cada uma das situações. A principal diferença observada está relacionada com o período das mensagens geradas, tendo-se obtido resultados distintos consoante o período das mensagens geradas seja inferior ou não a 200μs. Na figura 6.1 encontram-se os resultados de duas situações em

Condições (50000 pacotes)			**Sniffer Software**				
Período (μs)	Tamanho ($Bytes$)	Ocupação ($Mbps$)	Máximo (μs)	Mínimo (μs)	Média (μs)	Desvio Padrão(μs)	*Jitter* Absoluto (μs)
10	46	67	48872	2	15.39	335.62	48870
10	80	94	67542	2	13.78	373.52	67540
20	46	34	24786	2	21.76	158.31	24784
20	200	95	54927	2	22.78	276.42	54925
50	46	13	39159	2	50.90	188.11	39157
50	200	38	45161	2	56.32	283.38	45159
50	550	94	25654	2	56.43	249.81	25652
100	46	7	71817	2	101.46	334.54	71815
100	550	47	40805	2	105.51	280.21	40803
100	1150	95	117525	3	112.11	767.69	117522
200	46	3	280	121	200.03	2.37	159
200	1500	62	57007	80	202.78	291.44	56927
500	46	1	589	415	500.04	2.98	174
500	1500	25	53502	362	501.10	237.11	53140
1000	46	1	2969	6	1000.06	15.12	2963
1000	1500	12	50003	722	1001.82	272.29	49281

Tabela 6.2: Desempenho das ferramentas baseadas em *software* na realização do *time-stamping*.



(a) Captura baseada em *software*

(b) Captura baseada em *software*

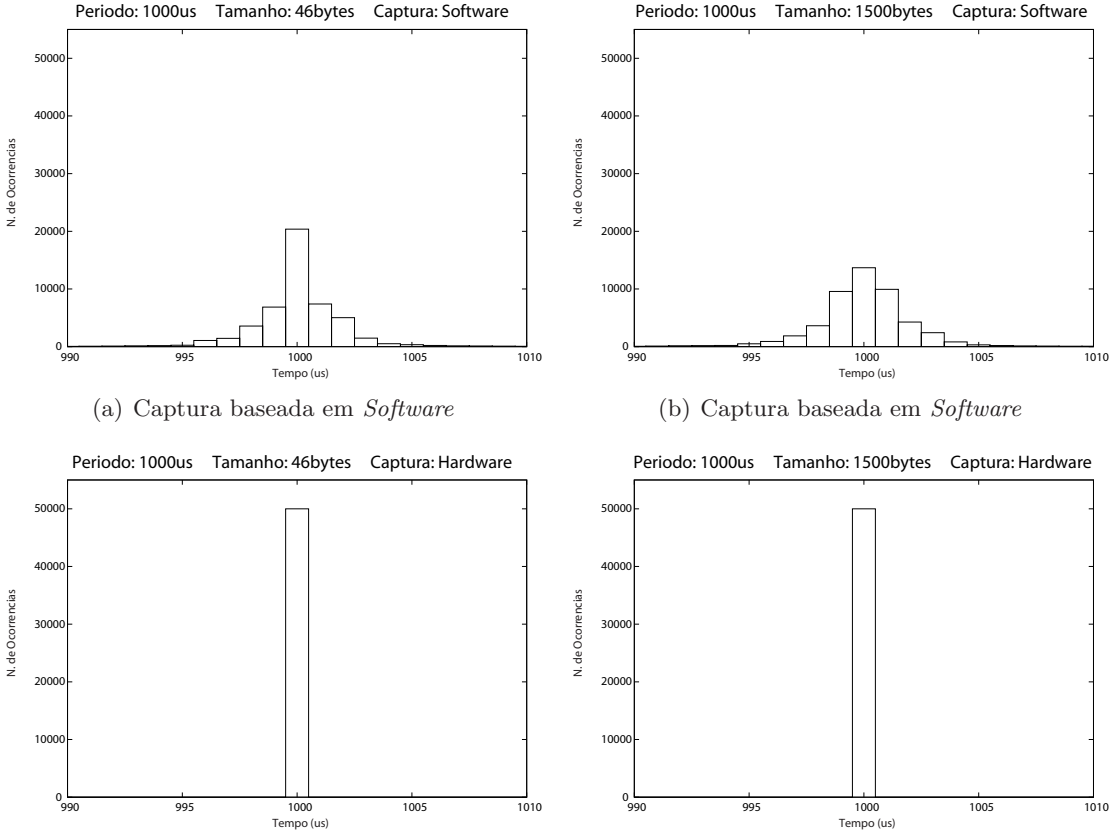
(c) Captura baseada em *Hardware*

(d) Captura baseada em *Hardware*

Figura 6.1: Resultados da realização de *time-stamping* em *hardware* e *software*.

Condições (50000 pacotes)			**Sniffer Hardware**				
Período (μs)	Tamanho ($Bytes$)	Ocupação ($Mbps$)	Máximo (μs)	Mínimo (μs)	Média (μs)	Desvio Padrão(μs)	*Jitter* Absoluto (μs)
10	46	67	10.09	9.99	10.02	0.035	0.10
10	80	94	10.09	9.99	10.02	0.035	0.10
20	46	34	20.09	20.00	20.02	0.034	0.09
20	200	95	20.09	19.99	20.02	0.034	0.10
50	46	13	50.09	50.00	50.02	0.034	0.09
50	200	38	50.09	50.00	50.02	0.034	0.09
50	550	94	50.09	50.00	50.02	0.034	0.09
100	46	7	100.09	100.00	100.02	0.033	0.09
100	550	47	100.09	100.00	100.02	0.033	0.09
100	1150	95	100.09	100.00	100.02	0.033	0.09
200	46	3	200.09	200.00	200.02	0.031	0.09
200	1500	62	200.09	200.00	200.02	0.031	0.09
500	46	1	500.09	500.00	500.02	0.022	0.09
500	1500	25	500.09	500.00	500.02	0.022	0.09
1000	46	1	1000.02	999.93	1000.01	0.022	0.09
1000	1500	12	1000.02	999.93	1000.01	0.022	0.09

Tabela 6.3: Desempenho das ferramentas baseadas em *hardware* na realização do *time-stamping*.



(a) Captura baseada em *Software*

(b) Captura baseada em *Software*

(c) Captura baseada em *Hardware*

(d) Captura baseada em *Hardware*

Figura 6.2: Impacto do tamanho das mensagens no desempenho das ferramentas de captura.

que o período das mensagens é inferior a 200μs. Por outro lado, na figura 6.2 encontram-se representadas duas situações em que o período é superior a 200μs, tendo sido variado o tamanho das mensagens. Estas permitem não só comparar a precisão dos resultados obtidos efectuando a captura em *hardware* e *software* mas também verificar o impacto do tamanho das mensagens no desempenho das ferramentas de captura.

Analisando os resultados verifica-se que a captura efectuada com *hardware* dedicado produz melhores resultados do que a captura utilizando um PC, em todas as situações. De facto, verifica-se que o valor do *jitter* nas capturas em *software* pode ser até 5 ordens de grandeza superior. Analisando os histogramas 6.1(a) e 6.1(b) verifica-se que as medições efectuadas em software se afastam imenso do que realmente se passa na linha de transmissão, podendo originar resultados enganadores. Na figura 6.1(b) verifica-se que apesar das mensagens apresentarem todas um período de 100μs os resultados obtidos indicam uma pequena minoria de mensagens associadas a este valor. Este acontecimento é comum a todas as situações em que o período das mensagens é inferior a 200μs, verificando-se que várias mensagens são processadas em simultâneo e podendo os resultados estar também corrompidos pela perda de algumas mensagens. A prova que as mensagens são geradas correctamente e que o problema se encontra na ferramenta de análise é apresentada na figura 6.1(d). Nesta é possível verificar que de facto todas as mensagens capturadas apresentam um período de 100μs. A justificação para os resultados obtidos com recurso a *software* tem a ver com o facto, já explicado anteriormente, do *time-stamping* das mensagens não ser efectuado no instante em que estas são recebidas mas sim no instante em que são processadas, estando os resultados relacionados com o tempo de processamento das mensagens. Isto é possível verificar na tabela 6.2 pois, em várias situações distintas, os resultados obtidos indicam um intervalo de tempo mínimo entre a recepção de duas mensagens igual a 2μs. Este tempo corresponde na realidade ao tempo de processamento de cada uma das mensagens. Por outro lado, com o aumento do período das mensagens, o desempenho do *Wireshark* melhora. Observando a figura 6.2 e a tabela 6.2 é possível verificar que os resultados reflectem uma situação que se encontra muito mais próxima do que realmente acontece no meio de transmissão. No entanto, apresentam ainda *jitter* fruto da ferramenta de captura, aumentando este com o tamanho das mensagens. Esta situação está relacionada com a necessidade do barramento PCI ser requisitado mais vezes para transmissão de cada mensagem, estando portanto sujeita a mais fontes de atraso e *jitter*.

Quanto à captura com recurso a *hardware* dedicado, é possível concluir que os resultados são independentes da taxa de utilização da rede, do tamanho e do período das mensagens. Tanto o valor do *jitter* como do desvio padrão se mantêm praticamente constantes em todas as situações. Verifica-se também que o valor médio do período é sempre superior àquele com que as mensagens são geradas, podendo esta situação estar relacionada com o facto de não ser utilizado o mesmo oscilador para contagem do tempo no emissor e no receptor, havendo uma pequena diferença na frequência de oscilação.

No que se refere ao valor do *jitter*, este apresenta um valor máximo de 100ns. Este valor contempla o *jitter* associado à captura e *time-stamping* das mensagens mas também o *jitter* introduzido na geração de tráfego. O TEMAC tem uma latência associada, tanto para recepção como transmissão de mensagens, que pode apresentar uma variação até 3 ciclos de relógio, ou seja, até 120ns. Se esta variação estiver completamente descorrelacionada do tamanho das mensagens, sendo a latência associada à recepção das mensagens completamente independente da latência associada à transmissão é possível que o *jitter* total atinja um valor máximo de 240ns. Isto considerando apenas o *jitter* associado a cada um dos TEMACs. Na prática, esta situação não se verificou, apresentado os resultados obtidos sempre valores

inferiores a um máximo de 100ns.

Teoricamente, considerando apenas o erro associado à recepção das mensagens, e se à variação na latência do TEMAC for adicionado o erro de 15ns associado à realização do *time-stamping*, é possível concluir que, no pior caso, a ferramenta permite efectuar medições com um erro máximo de 135ns.

6.3 Capacidade de Resposta da Ligação USB

Outro parâmetro que é importante analisar e do qual depende não só a duração da captura mas também a quantidade de dados capturados é a capacidade de transferir dados da FPGA para o PC. De seguida são apresentados alguns dos testes realizados para analisar a capacidade de resposta da ligação *USB* face à taxa de utilização da rede Ethernet.

Análise da duração da captura em ligações *half-duplex*

O primeiro teste consistiu na verificação da duração da captura com uma taxa de utilização da rede Ethernet muito próxima de 100Mbps. Para a mesma taxa de transmissão foram no entanto realizados vários ensaios quer usando pacotes de tamanho mínimo, quer pacotes de tamanho máximo. O número de pacotes capturados nos vários ensaios realizados encontra-se representado na tabela 6.4. Com excepção das situações assinaladas com *, nas quais a captura foi terminada pelo utilizador, em todas as outras a captura foi terminada devido a limitações na largura de banda da ligação USB que conduziram a um enchimento da memória destinada à informação de controlo.

Tamanho	Número de pacotes capturados (unidades)				
(*Bytes*)	Ensaio 1	Ensaio 2	Ensaio 3	Ensaio 4	Ensaio 5
46	135188	1440314	286882	720761	693742
1500	375878	696539*	544342	696435*	693742*

Tabela 6.4: Número máximo de mensagens capturadas com elevada taxa de utilização da rede Ethernet (*half-duplex*).

No caso das mensagens de tamanho mínimo foi possível capturar durante aproximadamente 10 segundos no ensaio 2. No entanto, no caso do ensaio 1, a duração da captura foi inferior a 1 segundo. Utilizando mensagens de tamanho máximo a captura foi na maior parte das vezes interrompida pelo utilizador após aproximadamente 80 segundos devido à grande quantidade de informação armazenada no PC.

Analisando os resultados obtidos verifica-se que estes apresentam uma grande variação ao longo dos vários ensaios realizados. Esta situação justifica-se devido ao facto de toda a transferência de informação via USB ser controlada pelo *host* USB do lado do PC. Devido às características multiprogramação associadas aos sistemas operativos já descritas, o PC não se encontra dedicado exclusivamente à recepção de dados via USB. Por este motivo, a taxa de transferência que se consegue alcançar na ligação USB depende entre outras coisas da taxa de utilização do processador e de latências associadas ao barramento PCI. Se o tempo que o PC demorar a atender outras tarefas (tempo de bloqueio do ponto de vista do *sniffer*) for demasiado longo e os dados recebidos via USB forem inferiores aos recebidos da rede Ethernet

Condições (500000 pacotes)			Pacotes sem campo de dados capturado (%)
Período (μs)	Tamanho ($Bytes$)	Ocupação ($Mbps$)	
10	46	67	0.12
10	80	94	0.30
40	200	48	0.09
40	350	78	0.37
40*	400	88	0.62
70	400	50	0.09
70	650	79	0.50
70*	800	96	0.78
100	600	51	0.14
100	900	75	0.57
100*	1150	95	0.96
130	800	52	0.22
130	1200	76	0.93
130*	1500	95	1.14

Tabela 6.5: Informação perdida em função da taxa de utilização da rede.

estes vão ficando armazenados na FPGA, até que esta atinge os limites da sua capacidade máxima de armazenamento.

Verifica-se ainda que apesar da taxa de ocupação da rede Ethernet ser igual nas duas situações obtém-se melhores resultados na situação em que são enviadas mensagens de tamanho máximo. Nesta situação o intervalo de tempo entre mensagens é maior e como tal a memória de controlo não enche tão rapidamente, havendo mais tempo para que os dados sejam transmitidos via USB. Para além disso, é possível limitar a quantidade de dados capturados de cada mensagem, sendo perdida informação.

Análise da limitação do conteúdo das mensagens capturadas

Para além da memória de controlo, também a memória de dados pode atingir os seus limites. Apesar desta situação não limitar a duração da captura, limita o conteúdo das mensagens capturadas. No pior caso, poderá não ser capturado qualquer byte, não sendo possível identificar a mensagem capturada. Para analisar esta situação foram realizadas várias experiências, tendo sido gerados 500000 pacotes em cada uma delas e nas quais se variou o intervalo de tempo entre a transmissão de mensagens e o seu tamanho. Na tabela 6.5 encontra-se a percentagem de pacotes dos quais não foi capturado qualquer byte. Daqui se verifica que a percentagem de mensagens sem qualquer byte capturado aumenta com o tamanho das mensagens. Esta situação acontece pois se a taxa com que as mensagens são recebidas for superior à taxa com que são enviadas, quanto maiores forem as mensagens mais rapidamente a memória destinada ao conteúdo das mensagens se esgota. Uma vez que poderá não ser importante capturar todos os bytes de cada mensagem, o *sniffer* foi também configurado de modo a capturar no máximo os primeiros 100 bytes de cada mensagem. Esta informação deverá na maior parte dos casos ser suficiente para identificar cada umas das mensagens capturadas. As condições em que foi testado nesta configuração encontram-se sinalizadas

Ocupação ($Mbps$)	Período (μs)	Tamanho ($Bytes$)	Período (μs)	Tamanho ($Bytes$)	Pacotes capturados (unidades)		
					Ensaio 1	Ensaio 2	Ensaio 3
193	40	450	100	1150	596458	1032764	241571
189	130	1500	130	1500	210737	559417	255171
189	10	80	10	80	4276	1970	513579
189	130	1500	10	80	562430	1834538	538408
189	10	80	130	1500	215376	323669	538251
118	100	700	100	700	419081	418667	413624
100	70	400	70	400	836116	866320	501813
99	40	200	100	600	207063	721683	1494837
95	40	200	40	200	595860	586438	577104
90	15	46	15	46	3474152	1739197	1725319
62	400	1500	400	1500	607461 *	693742 *	553567*
55	40	100	40	100	1015047	1016905	1495869
55	40	100	70	200	425564	3675323	3495832
54	70	200	70	200	721206	1026039	1732376
54	25	46	25	46	3643797	5832295 *	875066

Tabela 6.6: Número máximo de mensagens capturadas em função da taxa de utilização da rede (*full-duplex*).

com um * na tabela 6.5, tendo-se verificado uma redução para 0% no número de mensagens sem qualquer byte capturado.

Análise da duração da captura em ligações *full-duplex*

Finalmente, o *sniffer* foi testado numa ligação Ethernet *full-duplex*. Para isso foi gerado tráfego em ambos os sentidos perfazendo uma taxa de transmissão total próxima de 100%, 50% e 25%. Na tabela 6.6 encontram-se as condições em que as diferentes situações foram testadas, bem como o número de pacotes que foi possível capturar em cada um dos ensaios. É possível concluir mais uma vez que os resultados obtidos variam nos diferentes ensaios realizados, sendo a duração da captura fortemente dependente da capacidade do PC atender os dados disponíveis no controlador USB. Em algumas situações verificou-se que abrir uma nova janela da linha de comandos é suficiente para que a captura seja terminada. O facto de ser atendida outra tarefa e adiar a leitura dos dados presentes no controlador USB, é suficiente para que a capacidade das memórias da FPGA se esgote, terminando a captura.

Uma vez que a duração das capturas depende muito da ocupação do PC torna-se complicado caracterizar a capacidade de resposta do *sniffer*. Para além disso, o desempenho obtido poderá depender também do computador a que o mesmo se encontra ligado. Apenas as situações assinaladas com * na tabela 6.6 correspondem aos casos em que a captura não foi terminada por falta de recursos mas sim por opção do utilizador. É possível observar que a duração da captura não depende propriamente da taxa de utilização da ligação Ethernet mas sim do período das mensagens.

Captura	Mensagem	Máximo (μs)	Mínimo (μs)	Média (μs)	Desvio Padrão(μs)	*Jitter* Absoluto (μs)
Hardware	Trigger Message	1002.2	997.7	999.8	0.4	4.5
	Asynchronous Status	1005.9	995.4	999.8	0.9	10.6
Software	Trigger Message	1050	954	1000.1	2.6	96.0

Tabela 6.7: Análise do desempenho das ferramentas de captura: Caracterização das mensagens periódicas no protocolo FTT-SE.

6.4 Análise do Protocolo FTT-SE

O teste final consistiu na utilização do *sniffer* numa situação concreta envolvendo um sistema distribuído utilizando o protocolo FTT-SE. O *sniffer* foi então colocado entre o *switch* e o nó responsável pela aquisição de dados fazendo uso de uma câmara de vídeo. O conjunto de mensagens capturadas inclui a *Trigger Message*, enviada do Master para todos os slaves, a mensagem com informação do estado da fila de mensagens assíncronas (ASM), enviada do Slave para o Master "em simultâneo" com a *Trigger Message* e as mensagens com os dados capturados pela câmara e enviadas para a máquina responsável pela implementação do algoritmo de controlo.

A mesma captura foi também efectuada utilizando o *wireshark*, no entanto, devido à existência de uma única placa de rede no PC responsável pela captura, apenas se torna possível efectuar a captura num dos sentidos da ligação *full-duplex*, tendo neste caso sido capturadas apenas as *Trigger Messages*.

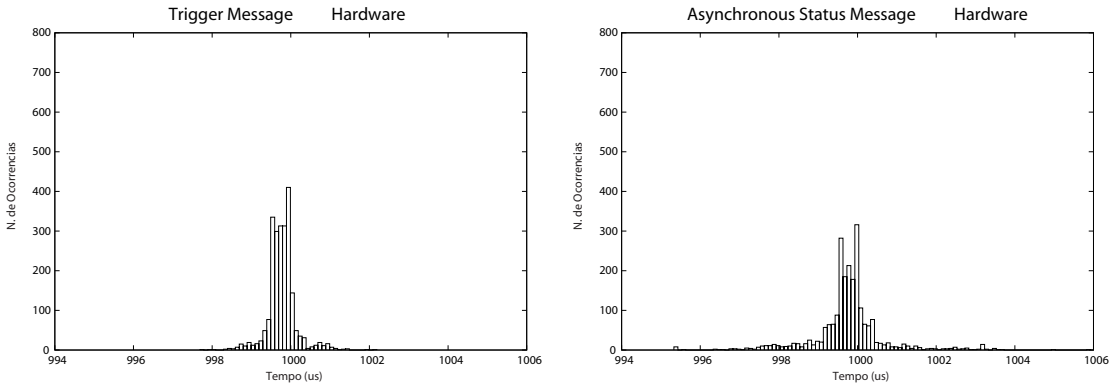
De forma a comparar os resultados obtidos, os dados foram exportados para o *octave*, tendo-se efectuado uma análise às mensagens periódicas capturadas. Os resultados obtidos encontram-se na tabela 6.7 e na figura 6.3.

Nesta situação em concreto as transmissões foram organizadas em ciclos elementares com a duração de 1 ms, sendo cada ciclo iniciado com o envio da *Trigger Message*. Na prática é possível verificar que este valor apresenta algum *jitter* devido ao facto do protocolo ser implementado em *software*. Da observação da tabela 6.7 e da figura 6.3 é possível concluir que os resultados das medições efectuadas variam consoante a forma como é efectuada a captura.

Os resultados obtidos efectuando a captura com base em software vêm afectados com o *jitter* inerente ao funcionamento do PC responsável pela captura e no qual é efectuado o *time-stamping* das mensagens. Para além disso, estas ferramentas de captura não permitem obter resoluções temporais inferiores a 1μs.

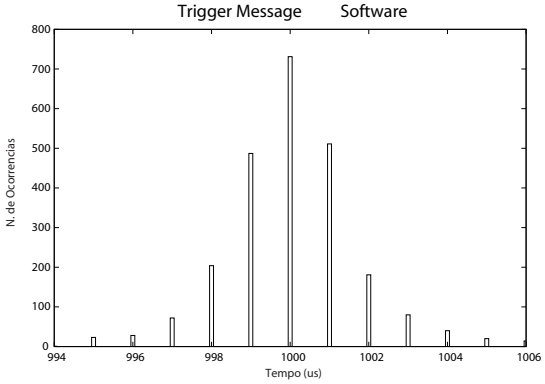
Efectuando a captura utilizando o *sniffer* construído obtém-se resultados que traduzem de uma forma muito mas precisa aquilo que realmente se passa no meio de comunicação. Para além disso, é possível verificar que a *Trigger Message* apresenta um *jitter* inferior à ASM. Isto deve-se ao facto da TM ser gerada pelo *Master* que apenas tem esta função. Por outro lado, as ASM são geradas nos *slaves* após sincronização com a TM anterior, tendo portanto mais fontes de atraso e *jitter*.

Nesta aplicação em concreto pode verificar-se algumas das potencialidades da ferramenta construída na análise de protocolos de tempo-real. Uma captura efectuada nas mesmas condições mas com o *Wireshark* resulta num valor de *jitter* associado à TM cerca de 20 vezes superior aquele que se obtém com a utilização de *hardware* dedicado. Desta forma,



(a) Trigger Message

(b) Asynchronous Status Message



(c) Trigger Message

Figura 6.3: Análise do desempenho das ferramentas de captura: *Jitter* nas mensagens periódicas utilizando o protocolo FTT-SE.

e nesta situação, é possível concluir o *Wireshark* introduz um erro nos resultados obtidos superior a 2133%.

Convém salientar que apesar de nem o *Wireshark* nem a ferramenta construída reflectirem exactamente o que se passa na rede de comunicação, o segundo permite efectuar uma análise mais exacta, estando os resultados menos afectados por erros de medição.

Capítulo 7

Conclusões

7.1 Resumo do Trabalho Realizado

O trabalho desenvolvido no âmbito desta dissertação visou a construção de uma ferramenta de análise de protocolos baseados na rede Ethernet, em particular o protocolo FTT-SE. Foi analisada a problemática do controlo em tempo-real em arquitecturas distribuídas e as limitações associadas às ferramentas de captura existentes baseadas em software. Neste contexto, foi desenvolvida uma ferramenta com características capazes de dar resposta aos requisitos típicos dos protocolos de tempo-real, sendo para isso desenvolvido um módulo de recepção, *time-stamping* e tratamento de mensagens baseado na linguagem de descrição de *hardware* VHDL e implementado em FPGA. De forma a que a introdução do módulo na rede seja o menos intrusiva possível foi também construído um TAP Ethernet que apresenta a vantagem de permitir capturar tráfego nos dois sentidos em ligações *full-duplex*. Recorrendo a uma aplicação baseada em software, para capturar a mesma informação, seria necessário utilizar um PC com duas placas de rede Ethernet. Para além disso, o *Wireshark* não permite efectuar a captura de dois dispositivos em simultâneo pelo que deveriam ser utilizadas duas aplicações, sendo gerados 2 ficheiros distintos.

Para efectuar a análise dos dados capturados foi definido e implementado um protocolo para transferência destes para um PC via USB. Desta forma, é possível apresentar os dados utilizando ferramentas *standard*, em particular o *Wireshark*, sendo possível tirar partido de todas funcionalidades que este oferece, desde a interpretação e descodificação de mais de 750 protocolos até à aplicação de filtros sobre as mensagens capturadas. Foram também desenvolvidos mecanismos de exportação dos dados para outras ferramentas como *matlab/octave* e folhas de cálculo que permitem a geração de gráficos, facilitando a análise visual dos resultados. Uma vez que os dados são exportados a partir do *Wireshark* torna-se possível fazer uma selecção dos dados que realmente interessa exportar, diminuindo desta forma a complexidade dos *scripts* de análise e consequentemente o seu tempo de execução que no caso de grandes quantidades de informação pode ser relativamente longo.

Finalmente, foi efectuado um conjunto de testes ao funcionamento global da ferramenta. Para isso foram feitas várias capturas utilizando quer o *sniffer* construído quer o *Wireshark*, permitindo assim efectuar uma comparação entre os resultados produzidos pelas duas. A informação capturada foi posteriormente exportada para o *octave*, no qual foi analisada, tendo sido gerados alguns gráficos com resultados. Na secção seguinte é efectuada uma análise final aos resultados alcançados.

7.2 Análise Final dos Resultados

Os resultados obtidos demonstram que a ferramenta construída permite medir diferenças temporais entre mensagens com uma precisão de 100ns e uma resolução de 10ns, conseguindo ter desempenhos semelhantes aos especificados por algumas ferramentas comerciais. Apesar do valor do erro máximo obtido ser inferior ao especificado por algumas ferramentas, estes são muitas das vezes valores péssimistas, verificando-se em todas as situações. Os resultados obtidos utilizando o *sniffer* construído são bastante melhores que os resultados alcançados utilizando aplicações baseadas em *software*, tendo-se obtido uma diferença de até 5 ordens de grandeza no *jitter* associado à medição do período de mensagens. Utilizando *hardware* dedicado o desempenho em termos temporais da ferramenta é independente do intervalo de tempo entre mensagens e do seu tamanho, o que não se verifica recorrendo à utilização de *software*. Nesta situação, os resultados são tanto melhores quanto maior for o intervalo de tempo entre mensagens e quanto mais pequenas estas forem. Para além disto, estas ferramentas permitem efectuar medições com uma resolução temporal de apenas 1μs apresentando um erro máximo da ordem dos milisegundos.

A principal limitação da ferramenta reside na transferência de informação da FPGA para o PC. A capacidade de transferir dados é fortemente dependente da capacidade do computador os receber. Este factor condiciona o tempo que cada captura pode durar pois se a capacidade de armazenamento temporário da FPGA se esgotar, a captura é terminada. Para que esta situação não aconteça, é necessário que o ritmo de transmissão com que os dados são enviados para o PC seja pelo menos igual ao ritmo de transmissão com que são recebidos da rede Ethernet. A realização de diferentes capturas com a mesma taxa de transmissão na rede Ethernet originou resultados bastante distintos, não sendo possível impor um limite abaixo do qual se garanta que a captura não seja terminada por falta de recursos. Uma forma de reduzir este problema passa por limitar a quantidade de dados capturados de cada mensagem. Esta situação permite uma menor utilização da memória destinada ao conteúdo das mensagens, evitando que sejam "descartadas" mensagens completas. Para além disso, a quantidade de informação a enviar via USB será menor e será atribuída uma maior percentagem da largura de banda para envio dos dados presentes na memória destinada à informação de controlo de cada uma das mensagens.

A arquitectura dos computadores de uso geral serve neste caso para justificar duas situações: o mau desempenho obtido utilizando aplicações baseadas em *software* e os resultados alcançados na transmissão de dados via USB. Estas máquinas são construídas para ser utilizadas nas mais diversas actividades, sendo por isso bastante flexíveis, no entanto, quando destinadas à execução de uma tarefa específica o seu desempenho não é optimizado. O que acontece na grande maioria das vezes é que existem múltiplas tarefas a serem executadas concorrentemente dando a ideia de simultaneidade. Por este motivo os recursos do sistema computacional são partilhados pela execução de diferentes tarefas, cabendo ao escalonador decidir qual a tarefa a executar quando houver várias prontas para execução. No entanto, a execução das tarefas pode muitas vezes ser interrompida, por exemplo, pela recepção de dados provenientes do teclado ou pela interface gráfica de gestão de janelas. No caso concreto da captura de mensagens utilizando um computador com uma placa de rede vulgar, o tratamento e *time-stamping* das mesmas não é efectuado ao nível da placa de captura mas sim ao nível do processador, sendo necessário, entre outras coisas, aguardar que o processador esteja disponível e requisitar o barramento PCI para transferência da informação. O tempo que decorre desde que as mensagens são recebidas até que cheguem ao processador é

variável estando sujeito a inúmeros atrasos. Para além disso, em determinadas situações são armazenadas várias mensagens na placa de rede sendo transferidas e processadas todas juntas, no entanto, se a transferência destas não for suficientemente rápida e a fila de mensagens da placa de rede encher as mensagens seguintes serão perdidas. Esta situação é semelhante à que acontece na recepção das mensagens via USB. Também aqui é necessário requisitar o barramento PCI para transferir os dados para a memória, estando as transferências sujeitas a atrasos variáveis. Apesar de neste caso o *timing* relativo da recepção de dados ser irrelevante é necessário que a informação seja recolhida de uma forma mais ou menos contínua da FIFO USB. As tarefas que bloqueiam a leitura de dados do controlador e o mecanismo de retransmissão, caso sejam detectados erros, originam a acumulação de dados nas memórias da FPGA, conduzindo ao esgotamento da capacidade de armazenamento do *sniffer* e consequente terminação da captura.

7.3 Trabalho Futuro

Apesar de todo o trabalho realizado, existem ainda alguns pontos que podem ser melhorados. De seguida são apresentados alguns aspectos passíveis de ser desenvolvidos e optimizados em desenvolvimentos futuros.

Transferência da FPGA para o PC

De forma a aumentar a capacidade de armazenamento temporário na FPGA, contornando desta forma aos tempos de bloqueio impostos pelo PC na recepção de dados via USB uma hipótese a considerar será a utilização de uma FPGA com um maior número de Block RAMs. Se se considerar a utilização da FPGAs da mesma família poderá ser utilizada uma Spartan3 XC3S5000 que disponibiliza 104 Block RAMs contra as 32 utilizadas neste trabalho. Caso se pretenda evoluir para outra família de FPGAs, aumentando o seu custo, poderá ser utilizada uma Virtex-4 (XC4VFX140), já com 4 controladores de acesso ao meio incluídos, permitindo poupar recursos, e que disponibiliza 552 Block RAMs. Outra hipótese a ter em consideração é a utilização de uma memória externa à FPGA. Isto invalida no entanto que a ferramenta possa ser implementada num único encapsulamento. As 32 Block RAMs encontram-se repartidas por 4 memórias de aproximadamente 16Kbytes, perfazendo um total de apenas 64Kbytes. Esta situação permitiria aumentar a capacidade de armazenamento para alguns Mbytes. A utilização, por exemplo, de uma memória DDR com capacidade para 32 Mbytes permitiria aumentar 512 vezes a capacidade de armazenamento da FPGA.

Estas soluções serão apenas adequadas se a taxa de transmissão média da rede Ethernet não exceder a taxa de transmissão média da ligação USB. Assumindo que não se pretende grandes capacidades locais de armazenamento, a única forma de resolver este problema passa pela substituição da ligação USB, devendo esta ser a principal consideração a ter em conta em desenvolvimentos futuros. Para isso deverá ser estudada uma alternativa à ligação USB, podendo ser considerada a utilização de uma interface *Gigabit* Ethernet (1000Mbps), FireWire (800Mbps) ou até mesmo uma ligação SATA (*Serial Advanced Technology Attachment*) (2400Mbps).

Criação de um *plug-in* para o *Wireshark*

Outro ponto em que é possível evoluir está relacionado com as ferramentas de captura e análise utilizadas do lado do PC. Neste contexto, seria interessante a criação de um *plug-in* para o *Wireshark* de suporte ao *sniffer* construído, integrando várias funcionalidades e permitindo "fechar" o sistema, sendo todo o *hardware* controlado a partir do *plug-in* criado. Este deveria, por exemplo, permitir iniciar e terminar a captura, sendo ao longo desta disponibilizada informação relativa ao número de pacotes recebidos bem como a duração da captura. Para além disso, era interessante configurar o número máximo de bytes de cada mensagem a ser capturados, especificar a captura ou não do campo FCS e definir o instante ou uma condição para iniciar a captura também a partir do *plug-in*.

Quanto às ferramentas de análise, seria importante que o *Wireshark* fosse capaz de descodificar todo o protocolo *FTT-SE*. Para além disso, seria também conveniente melhorar e integrar as ferramentas construídas com base no *octave* para análise de tráfego e geração de gráficos no próprio *plug-in*.

Posteriormente poderá também ser considerada a possibilidade da ferramenta gerar tráfego, permitindo desta forma analisar equipamentos de rede e efectuar medições de latências.

Ligação à Rede

Na implementação actual o *sniffer* é ligado à rede de comunicação utilizando um TAP Ethernet passivo. A forma como este foi concebido apresenta algumas limitações ao nível da atenuação dos sinais na linha de transmissão. Apesar de esta não se ter revelado significante nas ligações utilizadas em laboratório, poderá ter efeitos negativos se aplicado a cabos com maior dimensão ou menor qualidade. Por este motivo deverá ser estudada a possibilidade de construir um TAP activo, capaz de regenerar os sinais, diminuindo desta forma a atenuação causada nos mesmos pela introdução do *sniffer* na rede.

Precisão Temporal

A precisão temporal alcançada é limitada pelas características do controlador de acesso ao meio utilizado. A principal limitação reside na variação de 3 ciclos de relógio que a latência associada ao TEMAC apresenta. Uma forma de contornar este problema, reduzindo o erro associado às medições efectuadas passa pela utilização de um caminho paralelo ao TEMAC. Desta forma, para efectuar o *time-stamping* das mensagens seria necessário detectar o início da recepção de uma mensagem com base nos sinais provenientes do Phy através do barramento MII.

Apêndice A

Geração dos Núcleos de Propriedade Intelectual das Memórias

Nas figuras A.1 a A.6 são apresentados os vários passos seguidos para criação das memórias FIFO utilizando o *Fifo Generator*. Nestas é possível observar todas as características com que estas foram geradas. Neste caso em concreto as figuras correspondem à criação das memórias de controlo. A única diferença para as memórias destinadas ao armazenamento de dados é o tamanho do barramento de escrita. Na figura A.2 ao campo *Write Width* foi atribuído o valor 8 e ao campo *Write Depth* o valor 16384.

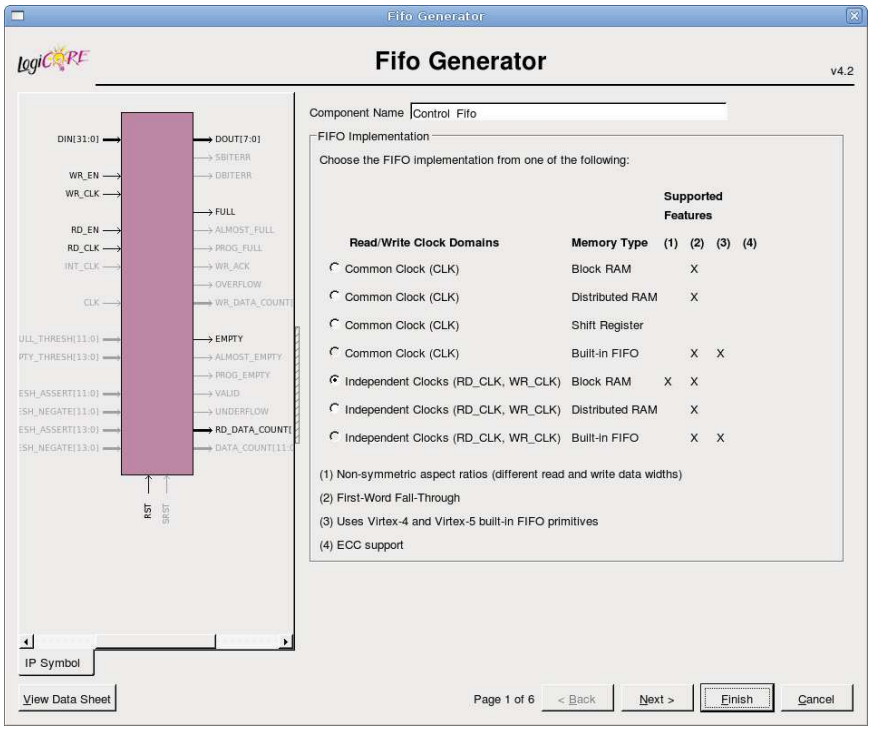


Figura A.1: Criação das memórias utilizando o *Fifo Generator* - passo 1 de 6.

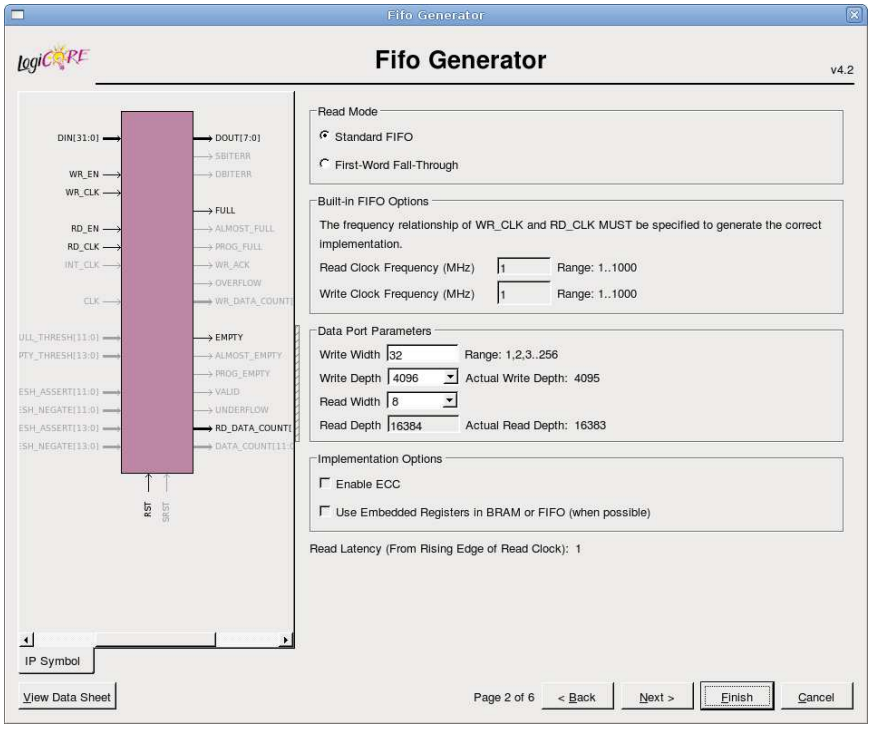


Figura A.2: Criação das memórias utilizando o *Fifo Generator* - passo 2 de 6.

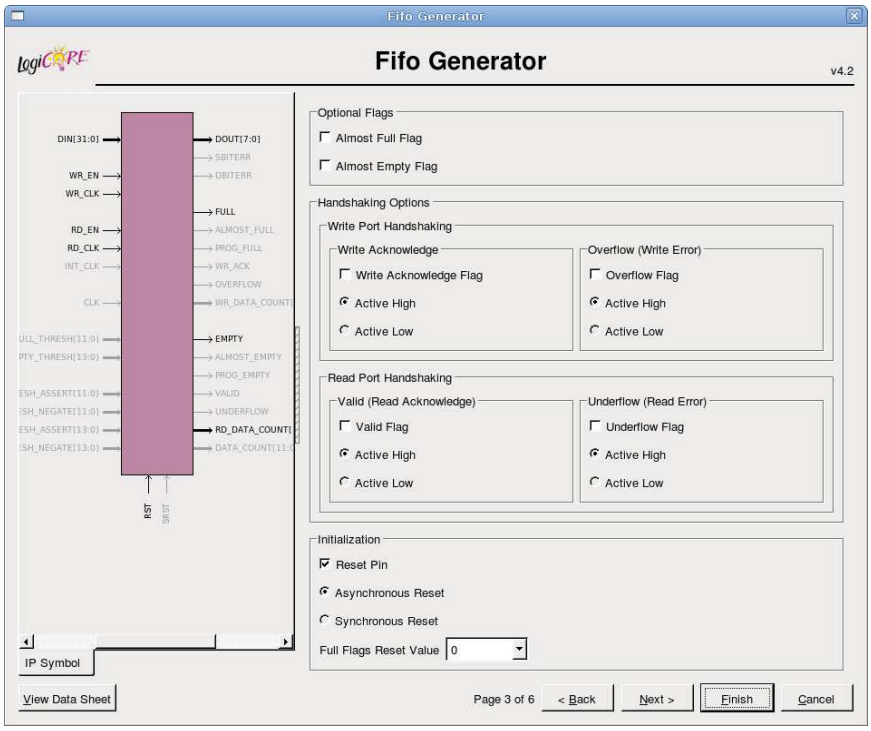


Figura A.3: Criação das memórias utilizando o *Fifo Generator* - passo 3 de 6.

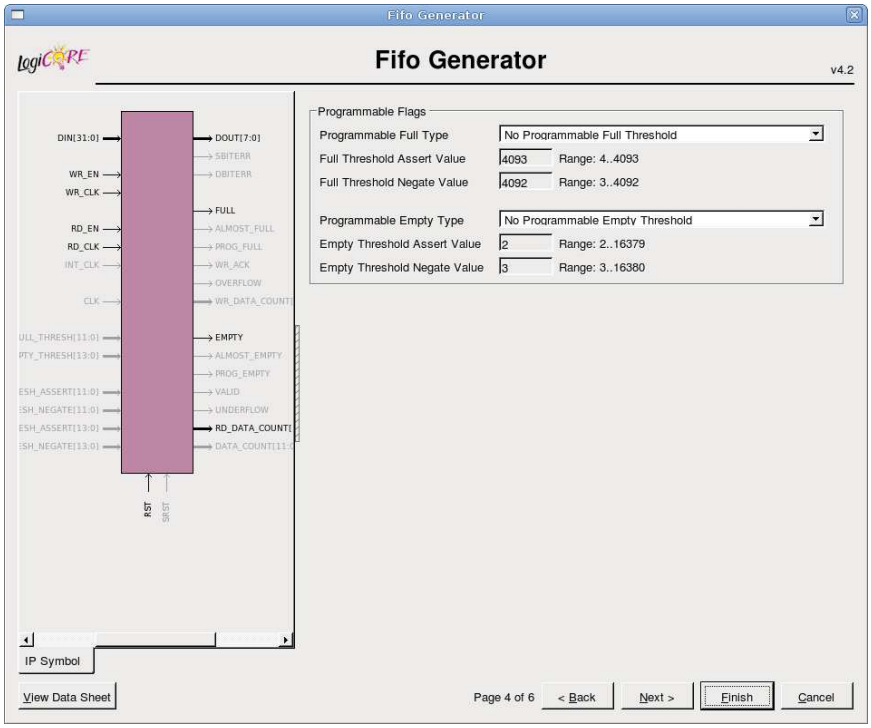


Figura A.4: Criação das memórias utilizando o *Fifo Generator* - passo 4 de 6.

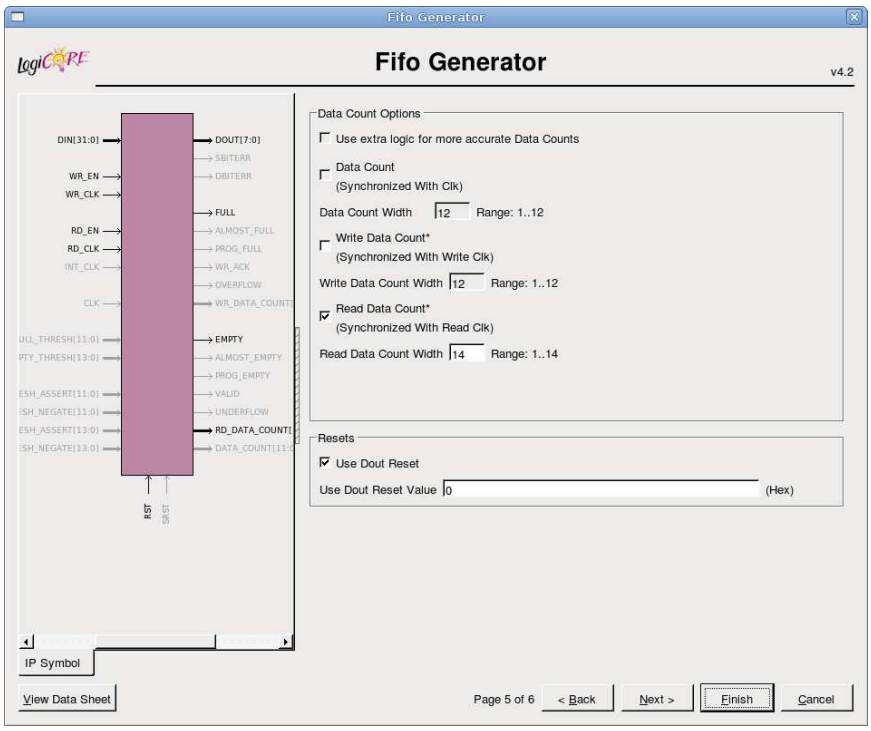


Figura A.5: Criação das memórias utilizando o *Fifo Generator* - passo 5 de 6.

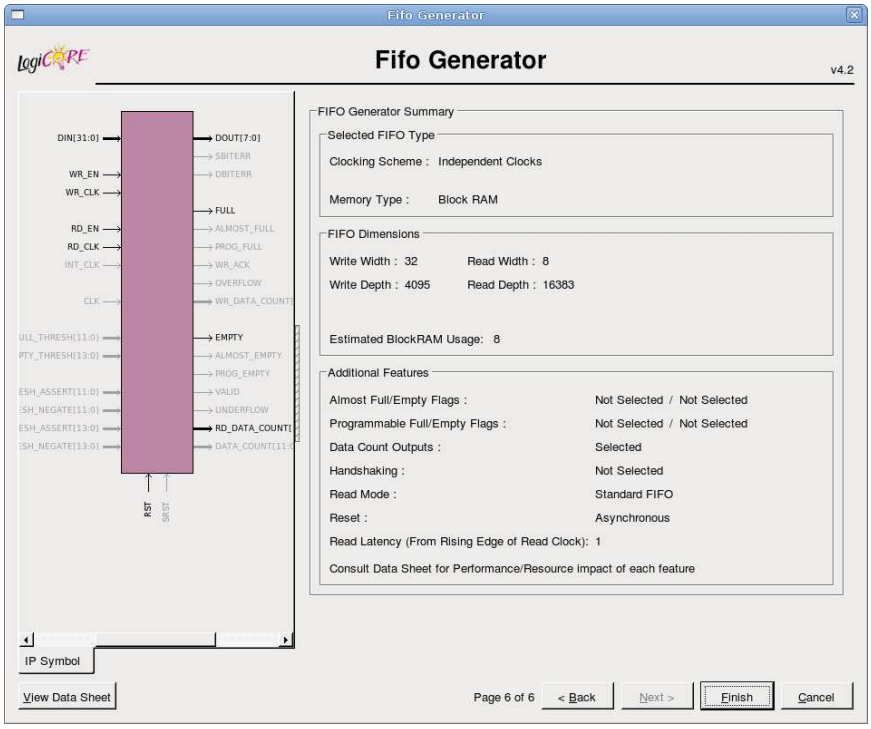


Figura A.6: Criação das memórias utilizando o *Fifo Generator* - passo 6 de 6.

Apêndice B

Geração do Núcleo de Propriedade Intelectual do Controlador de Acesso ao Meio

Na figura B.1 é possível observar as opções com que o controlador de acesso ao meio (TEMAC) foi gerado.

O barramento MII transfere os dados do Phy para o MAC a uma frequência de 25Mhz e utilizando palavras de 4 bits. Por outro lado, os dados são disponibilizados ao utilizador num barramento de 8 bits pelo que são actualizados a metade da frequência, ou seja 12.5Mhz, sendo necessário um sinal que permita sincronizar a leitura dos dados disponibilizados. Se a opção *clock enables* não for activada o controlador de acesso ao meio disponibiliza um sinal de relógio com esta frequência. No entanto, as linhas utilizadas especificamente para distribuição de sinais de relógio nas FPGAs são um recurso limitado. Por este motivo, à custa de alguma lógica adicional é possível poupar 2 linhas de distribuição de relógio. Activando a opção *clock enables* existe um sinal cujo o valor deve ser modificado em ciclos alternados do sinal de relógio de 25Mhz, permitindo efectuar uma distinção entre alguns dos seus flancos (figura 4.7). Quanto à opção *Management Interface* é utilizada para que o controlador gerado disponibilize uma interface que permite configurar o próprio controlador mas também aceder aos registos internos do Phy para sua configuração.

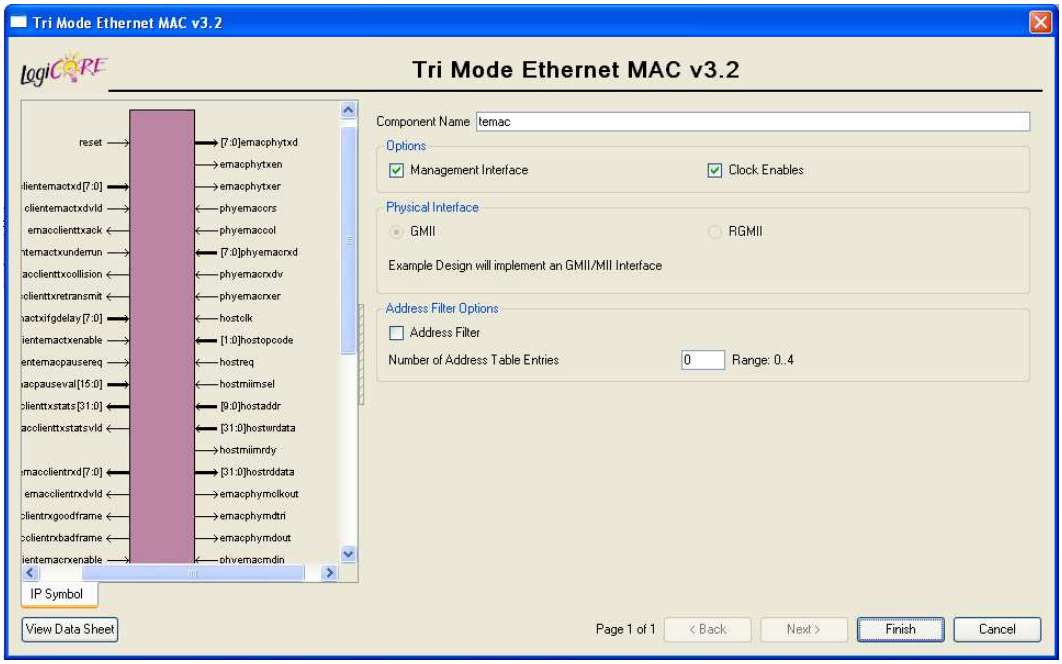


Figura B.1: Criação do bloco de controlo de acesso ao meio (TEMAC) utilizando o *Core Generator*.

Apêndice C

Lista de Acrónimos

API Application Programming Interface

ASIC Application Specific Integrated Circuit

ASM Asynchronous Status Message

ATM Asynchronous Transfer Mode

CAD Computer Aided Design

CAN Controller Area Network

CSMA/CD Carrier Sense Multiple Access with Collision Detection

CSV Comma Separated Values

DM Deadline Monotonic

DMA Direct Memory Access

EC Elementary Cycle

EDF Earliest Deadline First

ESP Electronic Stability Program

FCS Frame Check Sequence

FIFO First In First Out

FPGA Field Programmable Gate Array

FTT Flexible Time-Triggered Protocol

FTT-SE Flexible Time-Triggered protocol on Switched Ethernet

IEETA Instituto de Engenharia Electrónica e Telemática de Aveiro

ISE Integrated Synthesis Environment

JTAG Joint Test Action Group

LAN Local Area Network

LUT Lookup Table

MAC Medium Access Control

Mbps MegaBits Per Second

MDC Management Data Clock

MDIO Management Data Input/Output

MII Media Independent Interface

NIC Network Interface Card

OSI Open Systems Interconnection

PC Personal Computer

PCI Peripheral Component Interconnect

PLD Programmable Logic Device

PTP Precision Time Protocol

RM Rate Monotonic

RT Real-time

SATA Serial Advanced Technology Attachment

SFD Start Frame Delimiter

TDMA Time Division Multiple Access

TEMAC Tri-Mode Ethernet Media Access Controller

TM Trigger Message

USB Universal Serial Bus

Bibliografia

[ABRW91] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard realtime scheduling: The deadline monotonic approach. *The 8th IEEE Workshop on Real-Time Operating Systems and Software (RTOSS'91)*, pages 133–137, May 1991.

[AF05] P. Arlos and M. Fiedler. A Comparation of Measurement Accuracy for DAG, Tcpdump and Windump, 2005. Blekinge Institute of Technology (Sweden) [online] www.its.bth.se/staff/pca.

[Alt08] Corporation Altera. Altera webpage, January 2008. [Online] URL: http://www.altera.com.

[Anr08a] Company Anritsu, 2008. [Online] URL: http://www.eu.anritsu.com/.

[Anr08b] Company Anritsu. MD1231A, 2008. [Online] URL: http://www.anritsu.com.hk/downloads/files/MD1231Adatasheet.pdf?fileName=MD1231Adatasheet.pdf&fileID=1536&fileType=2.

[AP07] Luís Almeida and Paulo Pedreiras. Sistemas de tempo-real. [Online] URL: http://www.ieeta.pt/lse/str/str.htm, November 2007.

[APC05] APCON. Financial institution cuts analyzer costs by 77% equipment costs drop from 885*kto*200k, 2005. [Online] URL: http://www.arcadiz.com/content/assets/Suppliers/docs/apcon_ent_case_study_0605.pdf.

[Bos08] Portugal Bosch. Inovação para os automóveis: Sistemas de controlo electrónico de travagem bosch, February 2008. [Online] URL: http://www.bosch.pt/content/language1/html/734_4406.htm?section=CDAF31A468D9483198ED8577060384B3.

[But05] Giorgio C. Buttazzo. Rate monotonic vs. edf: Judgment day. real-time systems. January 2005.

[Can08] Can analyser, 2008. [Online] URL: http://www.cananalyser.com/.

[Ceb08] Marko Cebokli. Software for the UUUSB board (and CY7C68013 in general), 2008. [online]URL: http://lea.hamradio.si/ s57uuu/uuusb/uuusb_software.htm.

[CIA08] CIA. Can in automation - controller area network, 2008. [Online] URL: http://www.can-cia.de/index.php?id=170.

[Con08] Test Equipment Connection. Network analyzer : Anritsu md1231a, 2008. [Online] URL: http://www.testequipmentconnection.com/products/37104.

[Cyp07] Cypress. Semiconductor, January 2007. [online] URL: www.cypress.com.

[Dae05] Hauke Daempfling. Xilinx device families overview, December 2005. [Online] URL: http://ece.wpi.edu/~haukex/Xilinx_Tables.pdf.

[Dat08a] Datacom. Systems inc, choose the right network tap, March 2008. [Online] URL: http:// www.datacomsystems.com/solutions/choosing-network-taps.asp.

[Dat08b] Datacom. Systems inc, network tap products, March 2008. [Online] URL: http:// www.datacomsystems.com/products/network-taps.asp.

[Dat08c] Datacom. Systems inc, network tap solutions, March 2008. [Online] URL: http:// www.datacomsystems.com/solutions/overview.asp.

[Dec01] J-D Decotignie. A perspective on Ethernet-TCP/IP as a fieldbus. *Proceedings of the 4th FeT'2001 - International Conference on Fieldbus Systems and their Applications*, pages 138–143, November 2001.

[DFF+06] A. Depari, P. Ferrari, A. Flammini, D. Marioli, and A. Taroni. Multi-probe measurement instrument for real-time Ethernet networks. *Proc. of IEEE WFCS2006*, pages 313–320, June 2006.

[Eth07] Ethernet. [Online] URL: http://standards.ieee.org/getieee802/802.3.html, November 2007.

[Eur08] EurekaSpot. Network analyzer : Anritsu md1231a, 2008. [Online] URL: http ://www.eurekaspot.com/search/compare.cfm/LANANA/ANRI/MD1231A.html.

[Fin08] Corporation Finisar. Taps and splitters, March 2008. [Online] URL: http:// www.finisar.com/product_taps_8.

[Fon07] José A. Fonseca. Redes de comunicação em ambientes industriais. [Online] URL: http://www.ieeta.pt/ jaf/, November 2007.

[FTT07] FTT. Flexible Time-Triggered [Online] URL: http://www.ieeta.pt/lse/ftt/, September 2007.

[GR96] P. Gomes and R. Rijo. Pet: Uma ferramenta universal para avaliação de desempenho em sistemas distribuídos., 1996.

[HB98] B. Hedenetz and R. Belschner. Brake-by-wire without mechanical backup by using a ttp-communication network. *Proceedings of the 5th International Workshop on Real Time Networks (RTN'06)*, 1998. [Online] URL: http://www.vmars.tuwien.ac.at/projects/xbywire/projects/new-BBW.html.

[KDK+89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, pages 25–40, February 1989.

[Kis08] Kismet. Wireless, January 2008. [Online] URL: http://www.kismetwireless.net/.

[Kop97] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[Leo07] Patrick Leong. Ethernet 10/100/1000 Copper Taps, Passive or Active?, October 2007. [Online] URL: http://www.lovemytool.com/blog/2007/10/copper-tap.html.

[Lib07] Libpcap file format. [Online] URL: http://wiki.wireshark.org/Development /LibpcapFileFormat, December 2007.

[Lim08] Network Critical Solutions Limited. Critical taps, March 2008. [Online] URL: http:// www.criticaltap.com/catalogue.asp.

[Man08] InfiniStream Network Management, 2008. [Online] URL: http://www.empowered.ca/TestMgmt2006/NetworkGeneral/Infinistream.pdf.

[MB76] R. M. Metcalfe and D. R. Boggs. Ethernet: distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[MK85] M. Molle and L. Kleinrock. Virtual time csma: Why two clocks are better than one. *IEEE Transactions on Communications*, 33(9):919–933, September 1985.

[MPA06] Ricardo Marau, Paulo Pedreiras, and Luís Almeida. Enhanced ethernet switching for flexible hard real-time communication. *Proceedings of the 5th International Workshop on Real Time Networks (RTN'06)*, pages 45–48, July 2006.

[Nan05] Barry Nance. Software protocol analyzers and hardware probes, May 2005. [Online] URL: http://www.networkinstruments.com/assets/pdf/NetworkTestingLabs_2005.pdf.

[Net08] NetScout. Sniffer products, 2008. [Online] URL: http://www.netscout.com/products/.

[Oli07] Arnaldo Silva Rodrigues Oliveira. *Especialização e Síntese de Processadores para Aplicação em Sistemas de Tempo-real*. PhD thesis, Universidade de Aveiro, 2007.

[Opt07] OptiView. Link analyzer, November 2007. [Online] URL: http://www.flukenetworks.com.

[Opt08] Net Optics. Network taps, March 2008. [Online] URL: http://www. netoptics.com/products/default.asp.

[PA05] P. Pedreiras and L. Almeida. *Approaches to Enforce Real-Time Behavior in Ethernet*. CRC Press, 2005.

[Pac08] PackETH. ethernet packet generator, 2008. [Online] URL: http://packeth.sourceforge.net/.

[PAG02] Paulo Pedreiras, Luís Almeida, and Paolo Gai. The ftt-ethernet protocol: Merging flexibility, timeliness and efficiency. *Proceedings of the 14 th Euromicro Conference on Real-Time Systems (ECRTS.02)*, pages 134–142, 2002.

[PCSH99] G. Pardo-Castellote, S. Schneider, and M. Hamilton. Ndds: The real-time publish-subscribe middleware. *Real-Time Innovations, Inc, Sunnyvale, CA.*, August 1999. [Online] URL: http://www.rti.com/products/ndds/literature.html.

[PGAB05] P. Pedreiras, P. Gai, L. Almeida, and G.C. Buttazzo. FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. *Industrial Informatics, IEEE Transactions on*, 1(3):162–172, August 2005.

[PRO08a] Profibus combi-analyzer on usb, 2008. [Online] URL: http://www.procentec.com/profitrace2/.

[Pro08b] Ethernet Powerlink Protocol, 2008. [Online] URL: http://www.ethernet-powerlink.org/.

[SF03] Iouliia Skliarova and António B. Ferrari. Introdução à computação reconfiguravél. *Revista do DETUA*, 2(6), September 2003.

[Sno08] Snoop. Solaris Packet Sniffer, January 2008. [Online] URL: http://www.softpanorama.org/Net/Network_security/Sniffers/snoop.shtml.

[VC94] C. Venkatramani and T. Chiueh. Supporting real-time traffic on ethernet. *Proceedings of IEEE Real-Time Systems Symposium*, December 1994.

[WB04] Hans Weibel and Dominic Béchaz. IEEE 1588 Implementation and Performance of Time Stamping Techniques. *Conference on IEEE 1588*, September 2004.

[Wie08] Wolfgang Wieser. Home and Project Page of Wolfgang Wieser, 2008. [online] URL: http://www.triplespark.net/elec/periph/USB-FX2/software/.

[Wir08] Wireshark, January 2008. [Online] URL: http://www.wireshark.org/.

[X-B08] X-By-Wire. Safety related fault tolerant systems in vehicles, 2008. [Online] URL: http://www.vmars.tuwien.ac.at/projects/xbywire/.

[Xil06] Xilinx. *LogiCORE Tri-Mode Ethernet MAC v3.2 User Guide*, September 2006.

[Xil08a] Inc Xilinx. Chipscope pro: Optional real-time verification product in the ise design suite that provides on-chip debug at or near operating system speed, March 2008. [Online] URL: http://www.xilinx.com/ise/optional_prod/cspro.htm.

[Xil08b] Inc Xilinx. Ise foundation, March 2008. [Online] URL: http://www.xilinx.com/ise/logic_design_prod/foundation.htm.

[Xil08c] Inc Xilinx. Tri-mode ethernet media access controller (TEMAC), March 2008. [Online] URL: http://www.xilinx.com/products/ipcenter/TEMAC.htm.

[Xil08d] Inc Xilinx. Xilinx webpage, January 2008. [Online] URL: http://www.xilinx.com.

[Yok07a] Yokogawa. IP Traffic Load Generator/Analyzer - AE5511 Traffic Tester PRO, 2007. [online] URL: http://www.yokogawa.com/rd/pdf/TR/rd-tr-r00040-007.pdf.

[Yok07b] Yokogawa. Traffic Tester Mini AE5501, 2007. [online] URL: http://www.yokogawa.com/tm/data/ae5501/tm-ae5501_01.htm.